NPS ARCHIVE
2000.09
BROADSTON, R.

DUDLEY KNOX LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY CA 93943-5101

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

A METHOD OF INCREASING THE KINEMATIC BOUNDARY OF AIR-TO-AIR MISSILES USING AN OPTIMAL CONTROL APPROACH

by

Robert D. Broadston

September 2000

Thesis Advisor:	Robert G. Hutchins
Second Reader:	Hal A. Titus

Approved for public release; distribution is unlimited

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** September 2000	**3. REPORT TYPE AND DATES COVERED** Engineer's Thesis	
4. TITLE AND SUBTITLE: Title (Mix case letters) A Method of Increasing the Kinematic Boundary of Air-to-Air Missiles Using an Optimal Controls Approach			**5. FUNDING NUMBERS**
6. AUTHOR(S) Broadston, Robert D.			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited			**12b. DISTRIBUTION CODE**

13. ABSTRACT ***(maximum 200 words)***

Current missile guidance laws are generally based on one of several forms of proportional navigation (PN). While PN laws are robust, analytically tractable, and computationally simple, they are only optimal in a narrow operating regime. Consequently, they may not optimize engagement range, time to intercept, or endgame kinetic energy. The advent of miniaturized high speed computers has made it possible to compute optimal trajectories for missiles using command mid-course guidance as well as autonomous onboard guidance. This thesis employs a simplified six degree of freedom (6DOF) flight model and a full aerodynamic 6DOF flight model to analyze the performance of both PN and optimal guidance laws in a realistic simulation environment which accounts for the effects of drag and control system time constants on the missile's performance. Analysis of the missile's kinematic boundary is used as the basis of comparison. A missile's kinematic boundary can be described as the maximum theoretical range at which it can intercept a target assuming no noise in its sensors. This analysis is immediately recognizable to the warfighter as an engagement envelope. The guidance laws are tested against non-maneuvering and maneuvering aircraft targets and against a simulation of a cruise missile threat. An application of the 6DOF model for a theater ballistic missile interceptor is presented.

14. SUBJECT TERMS Missile Guidance Laws, Proportional Navigation, Optimal Control, Kinematic Boundary	**15. NUMBER OF PAGES** 226
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

A METHOD OF INCREASING THE KINEMATIC BOUNDARY OF AIR-TO-AIR MISSILES USING AN OPTIMAL CONTROL APPROACH

Robert D. Broadston
Lieutenant Commander, United States Navy
B.S.E.E., United States Naval Academy, 1984

Submitted in partial fulfillment of the
requirements for the degree of

ELECTRICAL ENGINEER

NPS ARCHIVE
2000 09
BRADSTON, E.
THESIS
B8023125
C.1

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

DUDLEY KNOX LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY CA 93943-5101

Current missile guidance laws are generally based on one of several forms of proportional navigation (PN). While PN laws are robust, analytically tractable, and computationally simple, they are only optimal in a narrow operating regime. Consequently, they may not optimize engagement range, time to intercept, or endgame kinetic energy. The advent of miniaturized high speed computers has made it possible to compute optimal trajectories for missiles using command mid-course guidance as well as autonomous onboard guidance. This thesis employs a simplified six degree of freedom (6DOF) flight model and a full aerodynamic 6DOF flight model to analyze the performance of both PN and optimal guidance laws in a realistic simulation environment which accounts for the effects of drag and control system time constants on the missile's performance. Analysis of the missile's kinematic boundary is used as the basis of comparison. This analysis is immediately recognizable to the warfighter as an engagement envelope. The guidance laws are tested against non-maneuvering and maneuvering aircraft targets and against a simulation of a cruise missile threat. An application of the 6DOF model for a theater ballistic missile interceptor is presented.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

I. INTRODUCTION 1
A. HISTORICAL BACKGROUND 3
B. GOALS AND BENEFITS 7

II. BACKGROUND 9
A. SIX DEGREE OF FREEDOM (6DOF) DYNAMICS 9
B. MISSILE MODELING 12
1. Airframes 13
2. Propulsion 16
3. Aerodynamics 17
a. Simplified 6DOF Model 17
b. Full Aerodynamic Model 21
4. Guidance, Navigation, and Control 23
a. Guidance 23
b. Navigation 24
c. Control 24
C. SIMULATION ENVIRONMENTS 25
D. GUIDANCE LAWS 27
1. Proportional Navigation 28
2. Velocity Compensated Proportional Navigation 29
3. Bang-bang 29
4. Differential Games 30
5. Augmented Proportional Navigation 31
E. TRACKING FILTER 32

III. GUIDANCE LAW TESTING 35
A. KINEMATIC BOUNDARY 35
B. TEST SCENARIOS 37
C. CANDIDATE GUIDANCE LAWS 39
D. NOISE STUDY 39

IV. COMPARISON AND ANALYSIS 41
A. PROPORTIONAL NAVIGATION LAWS 41
B. VELOCITY COMPENSATED PN LAWS 46
C. BANG-BANG 48
D. DIFFERENTIAL GAMES 50
E. AUGMENTED PROPORTIONAL NAVIGATION 53
F. NOISE STUDY 56
G. TBMD DEMONSTRATION 58

V. CONCLUSIONS AND FUTURE RESEARCH 61
A. CONCLUSIONS 61
B. FUTURE RESEARCH 62

APPENDIX A. SIMULINK® MODELS 63
APPENDIX B. MATLAB® CODE 93
APPENDIX C. SIMULATION DATA 185
A. PN (N'=5) 186
B. VCPN WITH CONSTANT GAIN 187
C. BANG-BANG 188
D. DIFFERENTIAL GAMES 189
E. APN WITH Λ=5 190
F. NOISY SEEKER, PN (N'=5) 191
G. FULL AERODYNAMIC MODEL 192
LIST OF REFERENCES 199
INITIAL DISTRIBUTION LIST 201

LIST OF FIGURES

Figure 2.1. Relationship of ABC, NED, and ECI Coordinate Frames. 10
Figure 2.2. AMRAAM and STANDARD models. Drawings to scale for comparison..... 13
Figure 2.3. Dimensions and forces on a tail-controlled missile. From [7]. 14
Figure 2.4. SM-2(ER) model with booster attached..... 15
Figure 2.5. AMRAAM and SM-2 (ER)..... 16
Figure 2.6. Variation of parasitic drag coefficient with Mach number. 19
Figure 2.7. Drag forces on the AMRAAM model for various load factors..... 21
Figure 2.8. Typical missile engagement geometry. From [13]..... 28
Figure 2.9. α–β–γ Filter Performance. 33
Figure 3.1. Kinematic boundary. The shooter is on the boundary pointing at the target at the start of the engagement..... 36
Figure 3.2. Comparison of a 5 meter warhead lethal radius to a MiG-29 aircraft. MiG-29 drawing is from [2]..... 37
Figure 4.1. Kinematic boundary comparison of proportional navigation laws vs. non-maneuvering, co-altitude target at 6,000 meters and Mach 0.83. 42
Figure 4.2. Kinematic boundary comparison of proportional navigation laws vs. maneuvering, co-altitude target at 6,000 meters and Mach 0.83. Target maneuver was a 6 g turn toward the missile at t_{go}=3 seconds. 43
Figure 4.3. PN law performance vs. non-maneuvering target. 44
Figure 4.4. PN law performance vs. maneuvering target. 44
Figure 4.5. Kinematic boundary comparison of PN vs. non-maneuvering and cruise missile targets. 45
Figure 4.6. Kinematic boundary comparison of VCPN laws vs. a non-maneuvering, co-altitude target at 6,000 meters and Mach 0.83. 47
Figure 4.7. Kinematic boundary comparison of the bang-bang law vs. a non-maneuvering, co-altitude target at 6,000 meters and Mach 0.83. 49
Figure 4.8. Kinematic boundary comparison of the differential games law vs. a non-maneuvering, co-altitude target at 6,000 meters and Mach 0.83. 51
Figure 4.9. Kinematic boundary comparison of the differential games law vs. maneuvering, co-altitude target at 6,000 meters and Mach 0.83. Target maneuver was a 6 g turn toward the missile at t_{go}=3 seconds. 52
Figure 4.10. Kinematic boundary comparison of APN vs. maneuvering, co-altitude target at 6,000 meters and Mach 0.83. Target maneuver was a 6 g turn toward the missile at t_{go}=3 seconds. 54
Figure 4.11. Kinematic boundary of APN and PN vs. cruise missile target. Azimuth resolution is 10 degrees. 55
Figure 4.12. Scatter plot of x and y miss distances for a noisy seeker. 56
Figure 4.13. Histogram of missile miss distances with a noisy seeker. 100 realizations. Probability of hit is 92%. 57
Figure 4.14. TBMD engagement by RIM-67 SM-2 (ER). Miss distance at intercept was 2.2 meters. 59
Figure 4.15. Interceptor and target velocity profiles for TBM demonstration. 59

Figure A.1. Simplified 6DOF model without tracking filter. 64
Figure A.2. Simplified 6DOF Aerodynamic Force Generator. 65
Figure A.3. Simplified 6DOF Drag Model. Function blocks are DRAGINDUCED.M and DRAGTHESIS.M. 66
Figure A.4. Flat Earth 6DOF missile dynamics. 67
Figure A.5. Internal missile dynamic model. SIXDOFDYN.M is the function block. 68
Figure A.6. Aerodynamic moment feedback 69
Figure A.7. Missile IMU and air data computer. Function blocks are Q2EULER.M and ALPHABETA.M 70
Figure A.8. Missile seeker model. 71
Figure A.9. Gimbal angles and rates. 72
Figure A.10. Range, range rate, and time-to-go. Function block is TGO.M. 73
Figure A.11. Target dynamics model. Function block is DYNAMIC3D.M. 74
Figure A.12. Target turn generator. Switch threshold is set to 3 seconds. 75
Figure A.13. Target and missile velocity computation 76
Figure A.14. Simplified 6DOF model with filter. 77
Figure A.15. Accelerometer. 78
Figure A.16. IMU with additional outputs for tracking filter. 79
Figure A.17. $\alpha-\beta-\gamma$ tracking filter. Function block is ABGFILTER.M. 80
Figure A.18. Full aerodynamic 6DOF model. 81
Figure A.19. Aerodynamic moment and force models. Function blocks are ALPHABETA.M, AEROFORCES.M, and AEROMOMENTS.M. 82
Figure A.20. Full aero model autopilot. 83
Figure A.21. Full aero model 6DOF equations. Function blocks are EQNFORCE.M, EQNMOMENT.M, EQNQUAT.M, and EQNPOS.M 84
Figure A.22. Rate gyros. 85
Figure A.23. Simplified 6DOF TBMD interceptor simulation. 86
Figure A.24. TBM thrust model. 87
Figure A.25. TBM target model. Function blocks are BALLISTDYN.M and GRAVITY.M. 88
Figure A.26. TBM seeker model. 89
Figure A.27. TBM missile model switch. Function block is MODELSWITCH.M. 90
Figure A.28. TBM Missile Dynamics. Function block is GRAVITY2.M. 91

LIST OF TABLES

Table 2.1. Missile Thrust Values 17
Table 3.1. Guidance Law Test Plan 39
Table B.1. Matlab® Source Code Listing 93
Table B.1. Matlab® Source Code Listing (continued) 94

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF SYMBOLS AND ABBREVIATIONS

Constant	Definition	Value
ω_x	*angular velocity of the Earth*	$7.292\times10^{-15}\ \frac{rad}{sec}$
g_0	*gravitational acceleration at Earth's surface*	$9.804\ \frac{m}{sec^2}$
GM	*earth – mass gravitational constant*	$3.986\times10^{14}\ \frac{m^3}{sec^2}$

CHAPTER II

Symbol	Definition	Components
v_B	*velocities (ABC frame)*	$\begin{bmatrix} v_x & v_y & v_z \end{bmatrix}^T$
ω_B	*angular velocities (ABC frame)*	$\begin{bmatrix} P & Q & R \end{bmatrix}^T$
q	*quaternions*	$\begin{bmatrix} q0 & q1 & q2 & q3 \end{bmatrix}^T$
p	*position vector (NED or ECI frame)*	$\begin{bmatrix} x & y & z \end{bmatrix}^T$
g_0'	*NED gravity vector*	$\begin{bmatrix} 0 & 0 & g_0 \end{bmatrix}$
F_B	*applied forces (ABC frame)*	$\begin{bmatrix} F_x & F_y & F_z \end{bmatrix}^T$
T_B	*applied moments (ABC frame)*	$\begin{bmatrix} T_x & T_y & T_z \end{bmatrix}^T$
J	*inertial matrix*	$\begin{bmatrix} J_{xx} & 0 & 0 \\ 0 & J_{yy} & 0 \\ 0 & 0 & J_{zz} \end{bmatrix}$
Ω_B	*body rate cross product*	$\begin{bmatrix} 0 & -R & Q \\ R & 0 & -P \\ -Q & P & 0 \end{bmatrix}$
Ω_E	*earth rate cross product*	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -\omega_x \\ 0 & \omega_x & 0 \end{bmatrix}$
Ω_q	*quaternion cross product*	$\begin{bmatrix} 0 & P & Q & R \\ -P & 0 & -R & Q \\ -Q & R & 0 & -P \\ -R & -Q & P & 0 \end{bmatrix}$

CHAPTER II

Symbol	Definition	Components
B_B	*NED rotation matrix*	$\begin{bmatrix} q0^2+q1^2-q2^2-q3^2 & 2(q1q2+q0q3) & 2(q1q3-q0q2) \\ 2(q1q2-q0q3) & q0^2-q1^2+q2^2-q3^2 & 2(q2q3+q0q1) \\ 2(q1q3+q0q2) & 2(q2q3-q0q1) & q0^2-q1^2-q2^2+q3^2 \end{bmatrix}$
μ, λ	*body latitude, longitude*	
B_G	*NED – ECI rotation matrix*	$\begin{bmatrix} \cos\mu & -\sin\mu\sin\lambda & \sin\mu\cos\lambda \\ 0 & \cos\lambda & \sin\lambda \\ -\sin\lambda & -\cos\mu\sin\lambda & \cos\mu\cos\lambda \end{bmatrix}$
B	*ABC – ECI rotation matrix*	$B_B B_B$
$g(p)$	*spherical earth gravity vector*	$-\frac{GM}{\|p\|^3}\begin{bmatrix} x \\ y \\ z \end{bmatrix}$
I_S	*specific impulse*	
$\dot{m}$	*propellant mass flow rate*	
C_{d0}	*parasitic (zero lift) drag coefficient*	
C_{di}	*induced drag coefficient*	
ρ	*atmospheric density based on ICAO standard atmosphere*	
V	*missile absolute velocity*	
S_{REF}	*reference area (missile cross sectional area)*	
C_N	*normal force coefficient*	
e	*wing efficiency relative to an elliptical planform*	
AR	*wing aspect ratio*	
C_M	*aerodynamic moment coefficient*	
d	*missile body diameter*	
α	*angle of attack (α used for pitch, β for yaw)*	
δ	*control surface deflection angle*	
C_N	*aerodynamic force coefficient*	
X_{XXX}	*missile dimensions see Figure 2.3 for definitions*	
v_x, v_y, v_z	*inertial frame target velocities*	
a_x, a_y, a_z	*inertial frame target accelerations*	
ω	*target turn rate*	

CHAPTER II

Symbol	Definition	Components
V_M, V_C	*missile and closing velocities. See Figure* 2.7	
$\sigma, \sigma_L, \dot{\sigma}$	*los angle, look angle, los rate. See Figure* 2.7	
n_c	*guidance law command acccceleration*	
N'	*navigation constant*	
A	*missile acceleration for bang – bang control*	
a_e, a_p	*evader and pursuer accelerations*	
c_e, c_p	*evader and pursuer energy constants*	
n_x, n_y, n_z	*missile command accelerations*	
Λ	*navigation constant*	
t_{go}	*time to go until intercept*	
I_3	3×3 *identity matrix*	
$\overline{0}$	3×3 *zero matrix*	
p_r	*relative position vector*	$[x_r \quad y_r \quad z_r]^T$
v_r	*relative velocity vector*	$[v_{xr} \quad v_{yr} \quad v_{zr}]^T$
A_T	*target inertial acceleration vector*	$[A_{Tx} \quad A_{Ty} \quad A_{Tz}]^T$
A_M	*missile body frame acceleration vector*	$[A_{Mx} \quad A_{My} \quad A_{Mz}]^T$

THIS PAGE INTENTIONALLY LEFT BLANK

EXECUTIVE SUMMARY

A six degree of freedom (6DOF) computer simulation of the AIM-120 AMRAAM has been developed to test the performance of various guidance laws using the kinematic boundary as a measure of effectiveness. Proportional navigation (PN) was used as the baseline for comparison. The effect of seeker noise on the PN law was studied.

A velocity compensated PN law was tested against an angles only PN law and demonstrated that the velocity compensation will improve performance, but not to the level of the full PN law.

A bang-bang law was tested as a continuation of earlier thesis work. This law performed poorly under the influence of drag, and would not be a candidate for use in a tactical missile.

A modified PN law derived from differential games theory was tested that had lower performance than the PN law.

An augmented PN law derived from optimal control theory was tested that had improved performance in the target's rear hemisphere and forward of 60 degrees relative to the nose of the target. This law did not improve the missile's performance against a cruise missile target.

Preliminary work to extend the 6DOF simulation to include aerodynamic control of the missile was completed with the simulation capable of limited operation. More work needs to be accomplished to bring this model to full capability.

The 6DOF model was used to demonstrate the engagement of a theater ballistic missile by a RIM-67 STANDARD II (ER) missile. The STANDARD intercepted the target at a range of 2.2 meters off the nose, well within lethal range of the interceptor warhead.

ACKNOWLEDGMENTS

The author wishes to thank Professors Robert G. Hutchins and Hal A. Titus for their guidance and assistance during his research. He would also like to thank his sister Mrs. Sharon Hall for her assistance in editing and formatting this thesis.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

The U.S. Navy's experience with Japanese *kamikaze* attacks in the closing months of the Second World War demonstrated the woeful inadequacy of anti-aircraft artillery (AAA) against a massed modern air threat. Even with radar-controlled guns and the massed firepower of dozens of ships, the *kamikaze* were able to inflict heavy damage on the fleet.

The advent of jet aircraft following the war exacerbated the aerial threat to surface units and changed forever the character of air-to-air combat. It was believed that the high speed and maneuverability of jet aircraft signalled the end of the dogfight and a requirement to engage targets at beyond visual ranges (BVR). The solution to both of these problems was some sort of guided missile.

There are a number of ways to guide a missile so that it hits a target. The three simplest guidance laws are beam rider, pursuit, and proportional navigation. Beam rider guidance is most useful for a surface-to-air missile (SAM) installation. The launcher must keep the target locked in a radar beam throughout the engagement while the missile steers along, or rides, the beam. This requirement ill suits beam riders for the dynamic environment of aerial combat. Pursuit guidance requires the missile to turn so that it continuously points at the target. The missile may use some characteristic of the target such as its infrared (IR) signature, semi-active radar from the launching platform, or onboard radar to determine the target's relative position. As the name implies, this guidance law is most effective when attacking from the rear hemisphere of the target, or when attacking a stationary target, and it performs poorly in the target's forward hemisphere.

Of the three basic guidance laws, proportional navigation (PN) is the most versatile, and therefore most frequently implemented. PN accelerates the missile laterally by an amount proportional to the angular rate of the line of sight from the missile to the target. PN or one of its various extensions or augmentations is the guidance law of choice in nearly all modern guided missiles. The reasons are simple. PN is:

- Cheap
- Robust
- Analytically tractable
- Successful

Optimal control theory promises to improve the performance of missile guidance systems. Optimal guidance laws, while the subject of extensive research, have yet to play a significant role in practical applications. Since optimal laws require an estimate of the target's position, velocity, and acceleration (its state), they require computing horsepower that has only been available in miniaturized form since the late 1980's. The computing requirements of optimal laws, and the successful extensions of PN laws have kept the optimal laws out of the mainstream, but modern, agile, stealthy aircraft and cruise missiles, and the growing need for theater ballistic missile defense (TBMD), have increased the interest in optimal guidance laws.

The remainder of Chapter I examines the historical background, our goals in pursuing this line of research, and its benefits. Chapter II establishes the theoretical background for the simulation environments and for the various guidance laws we examined. Chapter III describes our method of analysis using the kinematic boundary and our experimental procedures. Chapter IV presents experimental results and analysis. Chapter V presents our conclusions and suggestions for further research in this area.

A. HISTORICAL BACKGROUND

During the early 1950's, the development of guided missiles was a major program for the U.S. military. SAM's developed during this era include the Army's Nike family and Hawk, and the Navy's Terrier, Tartar, Talos, and Standard. The primary air-to-air missiles (AAM) of the day were the Raytheon Sparrow, developed for the Navy, and the Hughes Aircraft Falcon, developed for the Air Force. Both of these systems were complex radar-guided missiles (Falcon had an IR variant) and suffered from many developmental problems that would be familiar to systems engineers today.

While the engineers at Raytheon and Hughes were overcoming their technical challenges, a small team of scientists and engineers at the Naval Ordnance Test Station (NOTS) in China Lake, California, now the Naval Air Warfare Center Weapons Division (NAWCWPNS) began work on what would become one of the most successful AAM's in history. Sidewinder (AIM-9) began as an after work project on a non-existent and frequently purloined budget with no official standing [1]. In the view of the air power theorists of the day, the age of the dogfight was over, so why would there be a need for a short range dogfight missile? The Vietnam War would soon prove the theorists wrong and demonstrate the value of Sidewinder.

Sidewinder was designed from the beginning to be simple, reliable, rugged, and, above all, inexpensive. The motor, warhead, and fins were adapted from a stock five inch High Performance Air Ground (HPAG) rocket. The fins were modified with a mechanical device called a "rolleron" which minimized the missile's roll rate without the need for additional electronics [1]. Most of the design effort went into the guidance and

control section which was bolted on to the HPAG rocket as a unit, and incorporated several innovations, including:

- Torque balance servo control fins which provided the commanded control forces at all altitudes without complex electronics
- Single gyroscope seeker which integrated the IR sensor and directional gyro
- IR aiming reticles which reduced the missile's tendency to guide on the sun or clouds

The design was so simple that NOTS technicians would tell Air Force and Hughes personnel that the only test equipment they required was a flashlight and a Simpson meter [1]. While this may have been an exaggeration for psychological effect, it was not far from the truth. The first production Sidewinders cost the government $2,400 and by the third year of production, the price was down to $1,400 per missile [1]. Today's advanced Sidewinders cost in the tens of thousands of dollars. Compare this to over $300,000 for an AIM-120 AMRAAM. Sidewinder was such a successful design that it was copied wholesale by the Soviet Union as the K-13 (NATO AA-2 Atoll), and used as the basis for the Israeli Python [2], [1].

Sidewinder's guidance law is a form of PN using only line of sight angular rate and a fixed navigation constant or gain. This guidance law is suitable for a dogfight missile with a range on the order of 5.5 km (18,000 ft.), but not for longer ranges. The general PN law incorporates the missile's closing velocity with the target in the computation of the gain and must be provided a measurement of the range rate. This is the realm of the radar-guided missile.

The first radar-guided missiles in the U.S. inventory were the Air Force's Falcon (AIM-4) and the Navy's Sparrow (AIM-7). Both missiles used semi-active radar homing

(SARH) seekers. The launch aircraft must illuminate the target throughout the engagement for these missiles to guide successfully. Doppler processing of the illuminator's return from the target aboard the missile provides an estimate of the closing velocity. The need to continuously illuminate the target means that the launch aircraft must continue to close with the target during the engagement. This creates an obvious problem if the target's weapons have similar ranges to those of the launch aircraft.

Falcon enjoyed a long career, retiring in 1988. Sparrow is still in use today, and as NATO Sea Sparrow is the point defense missile system aboard many U.S. and NATO ships.

During the 1960's, Hughes began development of the missile that eventually became the AIM-54 Phoenix. Phoenix includes a strapdown inertial measurement unit (IMU) that allows its autopilot to steer the missile on course with periodic updating from a SARH seeker. In the terminal phase, the missile switches to an onboard active pulse Doppler radar. Finally, the missile has a simple data link with the AWG-9 radar aboard the F-14 launch aircraft that allows the aircrew to command the missile to perform several functions. All of these improvements permit the F-14 to simultaneously guide six missiles to different targets up to 176 km (110 miles) away.

Raytheon's AIM-120 Advanced Medium Air-to-Air Missile (AMRAAM) is the current generation of missile technology in the U.S. inventory. AMRAAM incorporates an IMU, a data link, and a pulse Doppler terminal seeker. Because its data link is more sophisticated than Phoenix, there is no need for a SARH seeker. In certain scenarios, AMRAAM is truly a "fire and forget" missile, using its IMU to fly to a point where the

active seeker can take over. Generally, AMRAAM is launched with an initial intercept solution programmed into the autopilot by the aircraft radar and mission computer. Once fired, the data link can update the autopilot with target position while the launch aircraft turns away or engages other targets. Once the terminal guidance seeker is activated, the missile is completely autonomous. AMRAAM has substantial onboard computer processing available and can employ advanced signal processing algorithms and guidance laws.

Proportional navigation can be shown to be an optimal solution under a set of limited conditions. Chief among these limitations is the assumption that the target does not maneuver during the engagement. This is clearly unrealistic, and there have been many extensions to the basic PN law to counter this limitation. Optimal control theory makes it possible to account for target maneuvers in the guidance law. This requires an estimate of at least the target's acceleration and in some cases the complete target state. A range of tracking filters including the Alpha-Beta-Gamma and Kalman filters is available to provide these estimates. Single chip microprocessors and digital signal processors have made it possible to implement these guidance laws in the limited volume of a missile's guidance section. Despite these developments and the potential advantages of optimal guidance laws, the practitioners have been slow to implement new designs. Some of this lag is due to the successful extension of the PN law, but much is due to the aversion of more experienced engineers for abandoning a technique that works in favor of techniques that have yet to prove themselves [3].

Modern agile aircraft like the MiG-29 and stealthy aircraft like the F-117 and F-22 may in some cases be able to defeat AAM's using PN laws. It is thought that optimal

and hybrid guidance laws may overcome the limitations of PN laws. Optimal laws may also increase the range at which cruise missiles can be engaged, and developments in differential games theory (a field of mathematical optimization) may help solve the TBM problem [4].

B. GOALS AND BENEFITS

The research presented in this thesis was motivated by three primary goals. First to create a set of 6DOF models for evaluating missile guidance laws, second to explore the use of the kinematic boundary as a measure of effectiveness (MOE) for evaluating the performance of the simulated missiles, particularly to compare optimal guidance laws with PN laws, and third to demonstrate an application of the models to a TBM interceptor.

Much of the literature in the missile guidance field involves the use of two-dimensional simulations. While such models are fairly simple to set up and analyze, and are not as computationally intensive as 6DOF models, they have difficulty simulating the effects of drag and aerodynamic control forces on the missile. Our goal was to create a simplified 6DOF model for guidance law development and testing, and a full aerodynamic model that would simulate both the aerodynamic control forces and the drag forces acting on the missile. The modular design of the Simulink® models makes it possible to test not only guidance laws, but autopilots, thrust profiles, and the effect of noise anywhere in the system on performance.

There are a number of ways to construct MOE's for the evaluation of a missile's performance. Controls engineers would compute a cost function based on the miss distance, control effort, and possibly time of intercept. While the number produced by

such a cost function is useful as a basis of comparison, to the layman it is simply a number. For the warfighter, the engagement envelope is of paramount importance. The kinematic boundary represents the maximum range at which the missile will achieve a hit when there is no noise in the system. It is a graphical representation of which guidance law has the best performance. If several points in the boundary are tested using noise, the mean effect of the noise can be calculated and its effect on the engagement envelope demonstrated. This information can then be used to determine if one guidance law is truly more effective than another. We have used the kinematic boundary as the MOE throughout the AAM simulations.

The final goal of this research was to provide a missile simulator that could be used in other research conducted for Navy TENCAP (Tactical Exploitation of National Capabilities) in the TBMD field.

II. BACKGROUND

A. SIX DEGREE OF FREEDOM (6DOF) DYNAMICS

Newton's laws for both translation and rotation describe the motion of a body in three-dimensional space. There are three axes for translation, x, y, and z, and three axes for rotation, longitudinal, lateral, and vertical, giving rise to displacement in roll, pitch, and yaw respectively. These are the six degrees of freedom. The coordinate frame for these dynamics is centered on the aircraft center of gravity (c.g.) and fixed to the airframe with the x-axis on the nose, y-axis on the right wing, and z-axis pointing down. It is called the aircraft-body centered or ABC frame. This is a rotating frame in inertial space and for objects in different ABC frames to interact; their motion must be transformed into an inertial frame.

For short ranges (< 200 km) the North-East-Down, or NED, frame is suitable. This frame assumes a flat earth, and reasonable altitudes so that gravity is a constant. A NED has its x-axis pointing north, y-axis pointing east, and its z-axis pointing down toward the center of the earth. An aircraft headed north in level flight will have pitch roll and yaw angles of zero degrees. A z-axis which points down seems counter-intuitive at first, but makes sense when one considers that this allows right hand turns to have an increasing heading as seen on a compass. We will use the NED or flat earth approximation for the air-to-air engagement simulations.

If the NED coordinate system were placed on the surface of the earth, it would become a rotating frame with the earth's angular velocity. For long ranges and ballistic missile work, one final translation to the earth-centered inertial or ECI frame is required. In this fixed frame the x-axis points at the vernal equinox or first point in Aries (which is

really in Pisces), the y-axis is 90 degrees to the east, and the z-axis extends through the North Pole. We will use the ECI frame for the TBM interceptor demonstration. Figure 2.1 shows the relationships of the three coordinate frames.

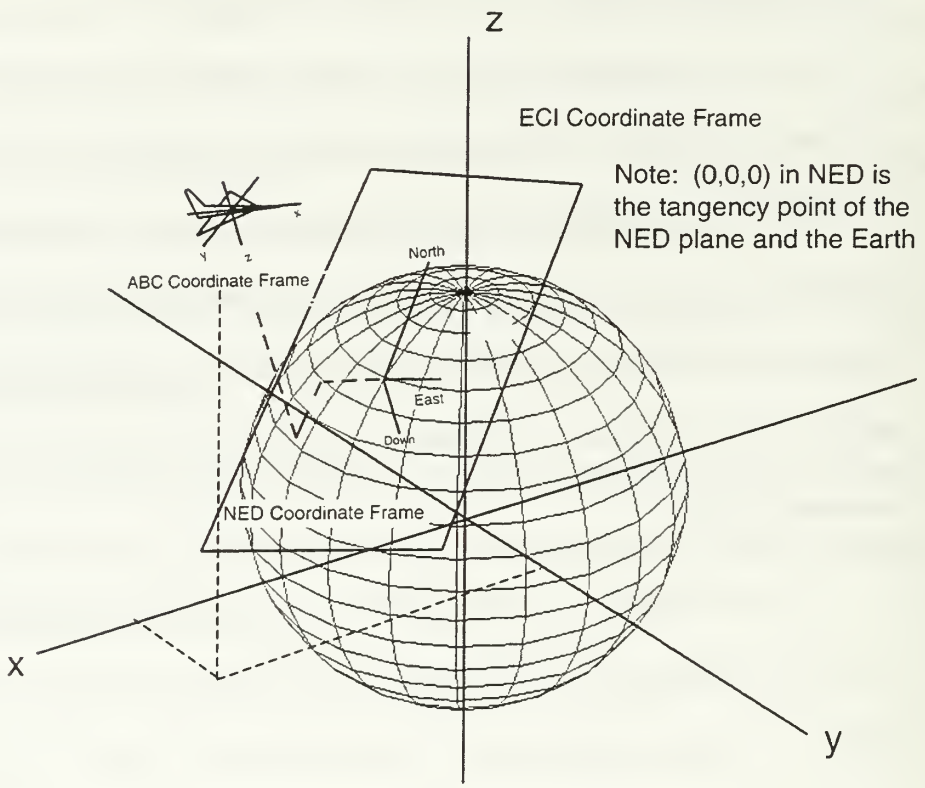


Figure 2.1. Relationship of ABC, NED, and ECI Coordinate Frames.

There are four vector equations which describe the dynamics of a body in three-dimensional space [5]. They are the force equation, the moment equation, the attitude equation, and the navigation equation. The equations shown below are for the flat earth approximation. The individual terms are defined in List of Symbols and Abbreviations.

$$
\begin{aligned}
\dot{v}_B &= -\Omega_B v_B + B_B g_0' + \frac{F_B}{m} && (force) \\
\dot{\omega}_B &= -J^{-1}\Omega_B J\omega_B + J^{-1}T_B && (moment) \\
\dot{q} &= -\frac{1}{2}\Omega_q q && (attitude) \\
\dot{p}_{NED} &= B_B^T v_B && (navigation)
\end{aligned}
\tag{2.1}
$$

The attitude equation can be computed using quaternions as shown here or using Euler angles. The Euler angle formulation involves a singularity in the rotation matrix (B_B) when the missile passes through the vertical that does not occur in the quaternion formulation. Since the STANDARD missile is fired from a vertical attitude, the quaternion formulation will be used throughout.

For the TBM interceptor demonstration, the round earth equations shown below in state space form are used. Note the addition of terms using Ω_E, which is the cross product matrix accounting for the Earth's rotation and the B matrix instead of B_B that rotates the ABC frame to NED coordinates, and then to ECI coordinates. Definitions of the individual terms are listed in the List of Symbols and Abbreviations.

$$
\begin{bmatrix} \dot{p} \\ \dot{v}_B \\ \dot{\omega}_B \\ \dot{q} \end{bmatrix} =
\begin{bmatrix}
\Omega_E & B^T & 0 & 0 \\
-B\Omega_E^2 & -\left(\Omega_B + B\Omega_E B^T\right) & 0 & 0 \\
0 & 0 & -J^{-1}\Omega_B J & 0 \\
0 & 0 & 0 & -\frac{1}{2}\Omega_q
\end{bmatrix}
\begin{bmatrix} p \\ v_B \\ \omega_B \\ q \end{bmatrix} +
\begin{bmatrix} 0 \\ Bg(p) \\ J^{-1}T_B \\ 0 \end{bmatrix}
\tag{2.2}
$$

These equations assume constant mass and a fixed center of gravity. Simulation of a missile that burns fuel and has a shifting c.g. as a result involves the addition of terms to the force and moment equations. For simplicity we have assumed a constant mass missile.

The MatLab® functions FLATEARTHDYN.M and SIXDOFDYN.M implement these equations in the SimuLink® models described below.

The equations of motion assume the motion takes place in a vacuum. As a result, there is no direct coupling between the force equation and the moment equation. A stable missile body with its c.g. forward of its center of pressure (c.p.) tends to act like a weather vane and align itself with the relative wind. In the simplified 6DOF model this is modeled by feeding back the angle of attack, which is the angle between the missile body and the velocity vector, and its derivative as a moment that steers the missile into the relative wind. The specifics of this feedback will be outlined below. The full aerodynamic model does not require this feedback as it generates the normal forces on the missile by generating a moment using control deflections and using the subsequent change in angle of attack to generate the forces.

B. MISSILE MODELING

The simulation environments are capable of modeling any missile the researcher chooses to represent. For this research, the AIM-120 AMRAAM, and RIM-67(ER) STANDARD II (SM-2) missiles were chosen. These weapons represent today's front line U.S. Navy technology.

The model dimensions have been simplified to comply with the supersonic aerodynamic models in Zarchan, and Blakelock, but are generally representative of the actual missiles [7], [8]. The performance specifications are also simplified and based on capabilities reported in the open source literature, and on engineering approximations. They are in no way intended to be representative of the actual capabilities of these

missiles. No official use U.S. Government or contractor proprietary documentation was used in the establishment of the model performance parameters.

1. Airframes

AMRAAM is a conventional missile design with fixed stub wings mounted forward on the missile body and controllable tail fins mounted aft. There are four wings and four fins mounted at 90-degree intervals around the missile body. Figure 2.2 shows the overall plan view of the missile, and the MatLab® file MISSILEDATA.M establishes the model's dimensions as required by Zarchan. The definitions of the dimensions used in Zarchan's equations are shown in Figure 2.3 [7].

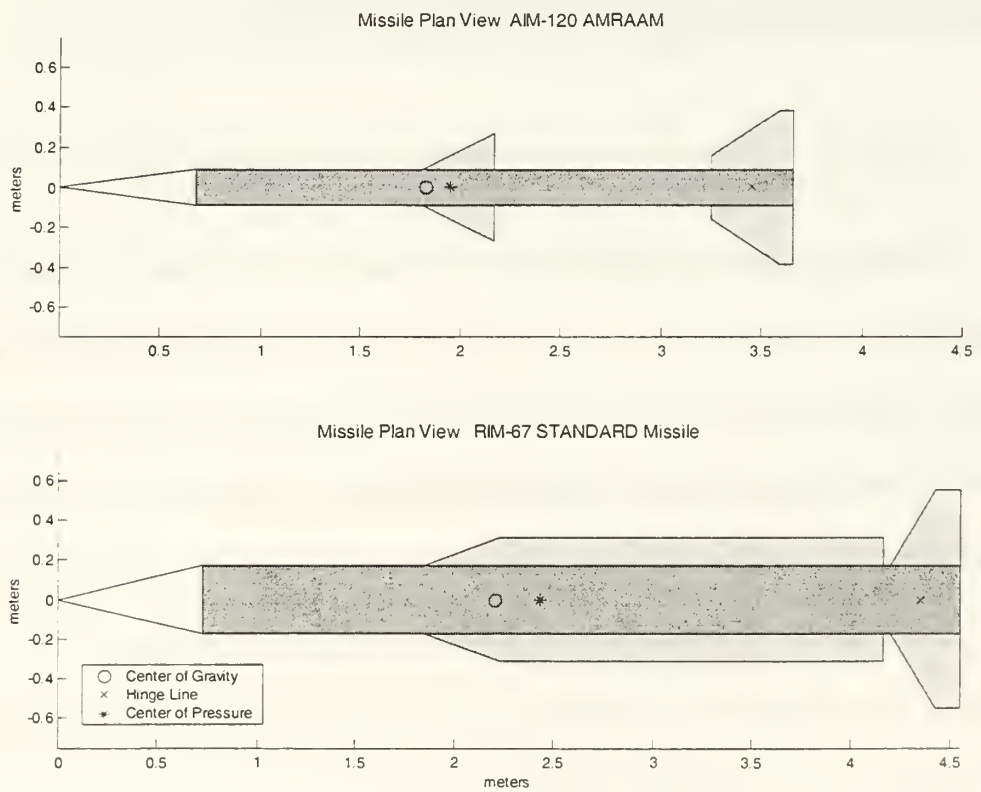


Figure 2.2. AMRAAM and STANDARD models. Drawings to scale for comparison.

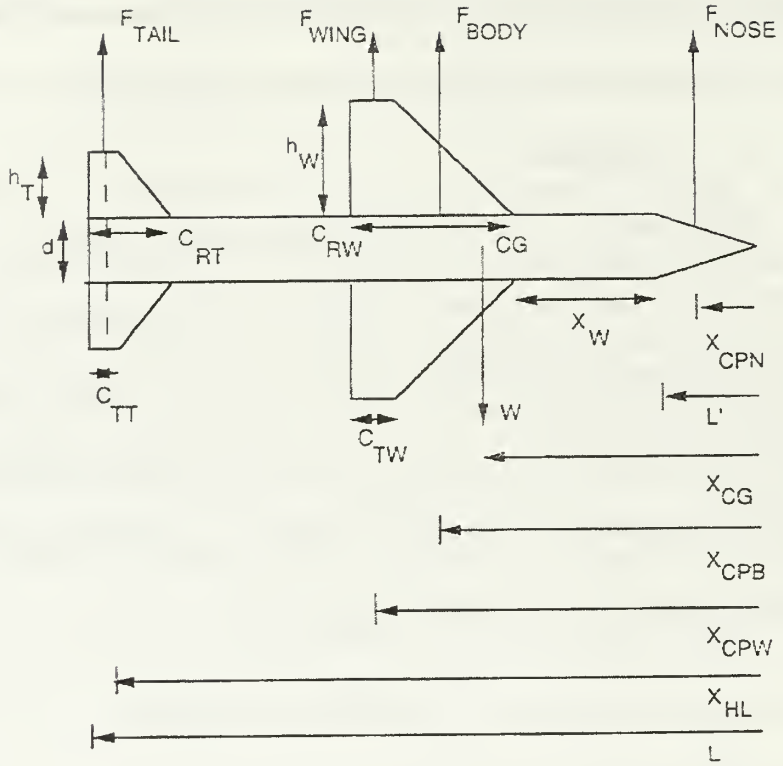


Figure 2.3. Dimensions and forces on a tail-controlled missile. From [7].

For the computation of the moments of inertia, the missiles are modeled as thin rods for the y and z-axes, and cylinders about the x-axis. The thin rod model was chosen, because the fins are not major contributors to the moment of inertia about the axes normal to the longitudinal axis, and the missile is much longer than its diameter so the contribution of the radius for the cylindrical model is minimal. The cylindrical model was chosen for the longitudinal axis because there is no moment of inertia for an infinitely thin rod about the longitudinal axis. Since the missiles are symmetrical, there are no cross terms in the inertial matrices.

The model for SM-2 is more complicated. The extended range version of the missile is equipped with a large booster with controllable tail fins. Figure 2.4 shows the

SM-2(ER) model with the booster attached. Note that the wings and tail fins for the missile forward of the booster have been modeled as a single wing with a length equal to the wing plus tail fin and an area equal to wing plus tail fin. MISSILEDATA4.M contains the dimensions for the missile in this configuration.

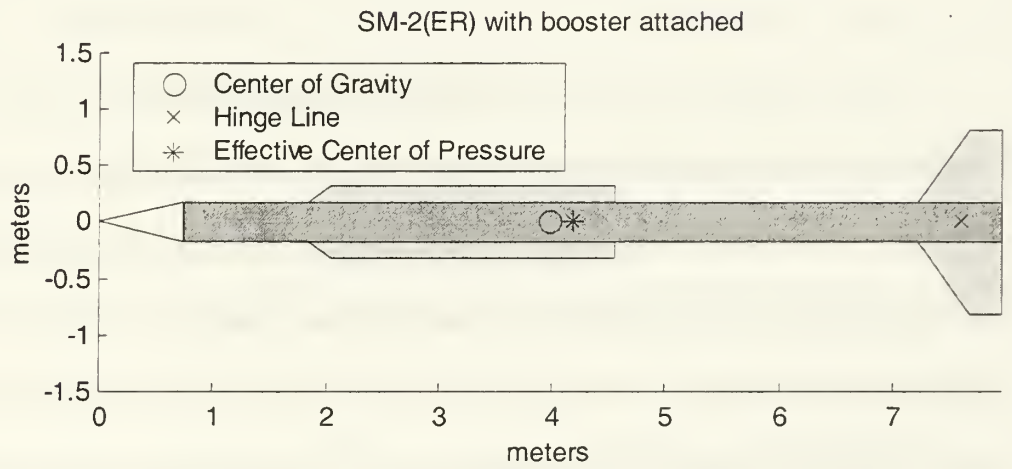


Figure 2.4. SM-2(ER) model with booster attached.

Once the booster stage falls away, the SM-2 looks like the second drawing in Figure 2.2. MISSILEDATA3.M contains the dimensions for the missile in this configuration. For comparison, line drawings of the actual missiles are shown in Figure 2.5.

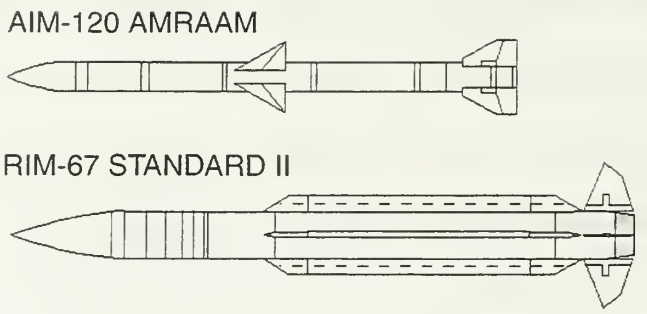


Figure 2.5. AMRAAM and SM-2 (ER).

2. Propulsion

Both AMRAAM and STANDARD use solid fuel rocket motors. The actual missiles use dual propellant grain motors that provide a relatively high value of thrust initially to accelerate the missile to speed quickly, and then a lower level of thrust to sustain speed throughout flight. For simplicity, the motors are modeled as single grain motors of intermediate thrust values.

Solid fuel motors used in military missiles must have a Department of Defense (DoD) Hazard Classification of 1.1 or 1.3 for use aboard ship [9]. According to Sutton, typical fuels of this type have specific impulses in a range of 180-270 seconds [9]. The thrust F produced by a rocket motor is given by:[9]

$$F = I_S \dot{m} g_0 \tag{2.3}$$

This equation assumes a constant propellant mass flow rate throughout the motor run.

Assuming propellant mass fractions of 50 percent for AMRAAM and SM-2 without its booster, and 80 percent for the SM-2 booster, with a six second burn time for AMRAAM and 10 seconds each for SM-2 and its booster yields the data shown in Table 2.1. The thrust values chosen for use in the simulations are within the range of

feasibility, and were chosen to accelerate the missiles to their maximum speed in a reasonable time.

Missile	Thrust Range (N) ($180<I_S<270$)	Simulation Thrust (N)
AMRAAM	23,062 – 34,594	23,000
STANDARD II	62,209 – 93,314	80,000
SM-2 BOOSTER	137,655 – 206,482	180,000

Table 2.1. Missile Thrust Values.

3. Aerodynamics

a. Simplified 6DOF Model

The aerodynamics for the simplified 6DOF simulation are modeled as a feedback path from the missile state vector. The ABC velocities are used to compute the pitch and yaw angles of attack (α and β) that are then differentiated and fed back as a proportional-differential (PD) controller to the torque input of the missile dynamics block (See Appendix B, Thesis1.mdl). This feedback loop models the missile's natural tendency to act like a weather vane when the lift and side forces change the velocity vector and hence the relative wind. The lift and side forces are generated by multiplying the guidance law command accelerations by the missile's mass.

The angle of attack response of the missile to a step input is similar to a second order response with a damped oscillation. This is also similar to the response of the full aerodynamic model to a step input on the control fins. The feedback gains were chosen to give the missiles a settling time of approximately 2.5 seconds, or an approximate first order time constant of 0.5 seconds.

Drag is modeled with two components, parasitic drag, that due to the misssile's shape and cross section, and induced drag, that caused by the generation of lift and side (normal) forces. The drag force D along the velocity vector is computed using the following equation [10].

$$D = (C_{d0} + C_{di})\rho \frac{V^2}{2} S_{REF} \tag{2.4}$$

Since the steady state angles of attack generated by this model are small, less than one degree, the small angle approximation has been used and the cosine of the angle of attack has been set to one for computing the component of drag along the x-axis of the missile.

C_{d0} is computed using typical values provided in [6]. The data were faired to a polynomial curve using MatLab®, and the function DRAGTHESIS.M is used to compute the parasitic drag in the model. Figure 2.6 shows the variation of C_{d0} with Mach number. The upper curve is the result of the increased drag caused by turbulence around the missile's tail when the thrust plume is absent.

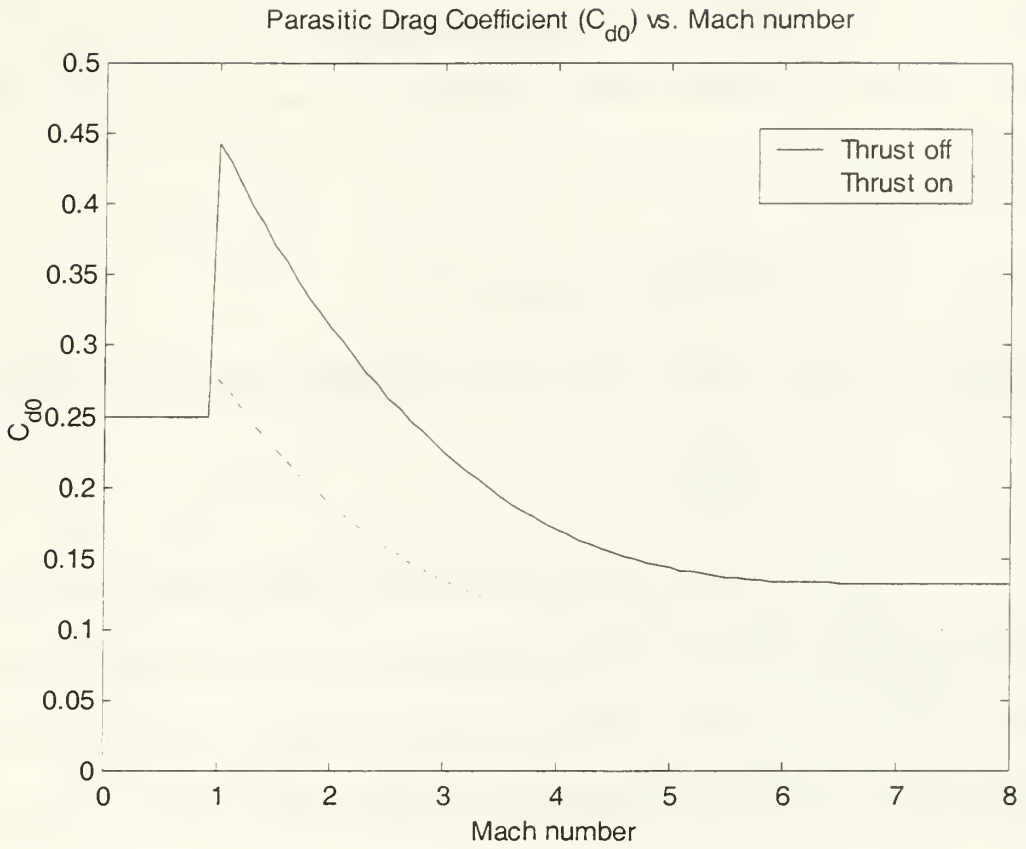


Figure 2.6. Variation of parasitic drag coefficient with Mach number.

C_{di} is computed in two regimes, subsonic, and supersonic. Normally, C_{di} is a function of angle of attack, but in this simplified model, the angle of attack values are not realistic, and therefore, a different approach is required.

For subsonic flight, C_{di} is computed as the applied normal force in g's times the maximum value of C_{d0} in subsonic flight. This crude approximation only affects the missile for very short periods of time as it is subsonic only at launch and perhaps at the very end of an engagement.

In supersonic flight, a more accurate approximation based on the normal forces is used. The normal force coefficient C_N is computed using the following equation:

$$C_N = 2\frac{F_N}{\rho V^2 S_{REF}} \tag{2.5}$$

where F_N is the applied normal force. C_{di} is then computed using the following [10].

$$C_{di} = \frac{C_N^2}{\pi e (AR)} \tag{2.6}$$

Since there are normal forces on both the y and z-axes, C_{di} is computed for each axis and the results are added to produce the value of C_{di} used in Equation 2.4 above. Figure 2.7 shows the parasitic and induced drag forces on the AMRAAM model for a missile in level flight executing a variety of turns at load factors up to 30 g's.

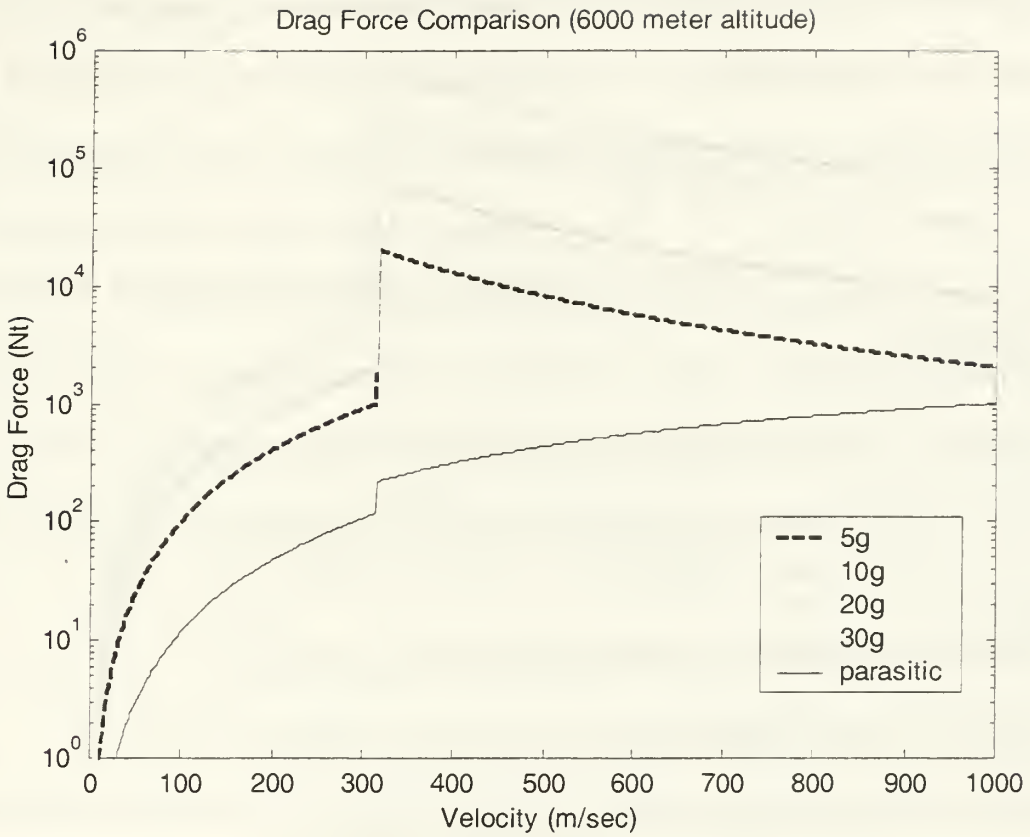


Figure 2.7. Drag forces on the AMRAAM model for various load factors.

b. Full Aerodynamic Model

The full aerodynamic model follows the development in Zarchan for generation of both the aerodynamic moments and forces. Moments are generated by the deflection of the appropriate control surfaces (rudder or elevator). The simulated missile flies in a vertical attitude with the elevator surface horizontal and the rudder surface vertical. AMRAAM flies in a "cross" configuration with the tail surfaces at 45 degree angles to the vertical for ease of loading and carriage aboard aircraft. Modeling this involves a more complicated autopilot and the change to a vertical attitude does not

materially affect the simulation. The aerodynamic moment T caused by a control surface deflection is given by:[7]

$$T = C_M \rho \frac{V^2}{2} \frac{S_{REF}}{d} \tag{2.7}$$

C_M is a function of the angle of attack and the control deflection and is given by:[7]

$$\begin{aligned} C_M = \; & 2\alpha\left(X_{CG} - X_{CPN}\right) + \frac{3}{2}\frac{S_{PLAN}\alpha^2}{S_{REF}}\left(X_{CG} - X_{CPB}\right) \\ & + 8\frac{S_W \alpha}{\beta S_{REF}}\left(X_{CG} - X_{CPW}\right) + 8\frac{S_T\left(\alpha + \delta\right)}{\beta S_{REF}}\left(X_{CG} - X_{HL}\right) \end{aligned} \tag{2.8}$$

Where β is a normalized speed for supersonic travel given by:[7]

$$\beta = \sqrt{Mach^2 - 1} \tag{2.9}$$

The normal force F_N on a body is given by:[7]

$$F_N = C_N \rho \frac{V^2}{2} S_{REF} \tag{2.10}$$

C_N is again a function of the angle of attack and the control deflection and is given by:[7]

$$C_N = 2\alpha + \frac{3}{2}\frac{S_{PLAN}\alpha^2}{S_{REF}} + 8\frac{S_W \alpha}{\beta S_{REF}} + 8\frac{S_T\left(\alpha + \delta\right)}{\beta S_{REF}} \tag{2.11}$$

The equations for C_N and C_M given above are valid for the supersonic regime. No such approximation based on missile dimensions exists for the subsonic regime. For subsonic speeds, the coefficients normally are determined empirically using wind tunnel or computed fluid dynamics data. Since these data were not available, C_N and C_M in the subsonic range are modeled as linear functions of the angle of attack and

control deflection [10]. Equations 2.7 and 2.10 are then used to generate the moments and forces. The values chosen for the coefficients are therefore arbitrary, but as with the drag model above, since the simulation spends very little time in the subsonic regime, the effects of this approximation will be minimal.

The drag model is quite different from the simplified 6DOF model. Parasitic drag is computed in the same fashion as above. Subsonic induced drag is computed using Equation 2.6, because the model now explicitly calculates C_N. Supersonic induced drag follows an approximation given in Anderson [10].

$$C_{di} = 4\frac{\alpha^2}{\beta} \tag{2.12}$$

Since the angles of attack are generally greater than one degree, the induced drag force due to each normal force is computed separately, and its component along the longitudinal axis of the missile is computed before being added to the other component. The parasitic drag force is multiplied by the cosines of both angles of attack to determine its longitudinal component.

According to Stevens and Lewis, once the moments and forces have been determined, the 6DOF equations are solved in the following order [5]:

- Force and moment equations
- Attitude equation
- Navigation equation

4. Guidance, Navigation, and Control

a. Guidance

Guidance laws are implemented as Matlab® functions which compute the command accelerations (n_c) for both lateral and vertical guidance. The inputs to the

guidance law are provided by the seeker head, which computes target range, range rate, azimuth, elevation, and angular rates from the actual target and missile state vectors. Measurement noise can then be added to any of the six output channels to study its effect on guidance law performance

Guidance laws for the simplified 6DOF model must also generate the applied force on each axis for the computation of the drag forces. Simulink® generates "algebraic loop" errors when the forces are fed back from the input of the "Missile Dynamics" block (Figure B.1). Guidance laws for the full aerodynamic model do not require this additional output.

Guidance laws requiring a tracking filter incorporate the filter's estimate of the target state and missile body frame accelerations as additional inputs.

b. Navigation

The inertial measuring unit (IMU), air data computer (ADC), accelerometers, and rate gyros provide navigation data to the missile simulation in the form of Euler angles, missile total velocity, acceleration, position, angles of attack, and body axis rotation rates. The IMU is mounted at the missile's c.g., thus simplifying the calculation of the accelerometer data. Although this research assumed a noise-free navigation system, noise sources could be added to any of the output channels to study the effect on performance. In particular, the effect of navigation system noise on the tracking filter could be studied.

c. Control

The simplified 6DOF model does not require an autopilot, since the guidance law command accelerations are directly converted into aerodynamic forces. For

the full aerodynamic model, it is necessary to convert the command accelerations into control deflection angles. For this purpose, an autopilot for tail-controlled missiles presented in Blakelock was adapted for use [8]. Blakelock's autopilot contains feedback loops for a missile which does not guide during boost, and to correct for accelerometers which are not at the c.g. These loops have been deleted in this model.

C. SIMULATION ENVIRONMENTS

Two distinct simulation environments were developed for this research. The simplified 6DOF model was designed initially for the purpose of developing and testing guidance laws prior to using them in the full aerodynamic model. Problems with the non-linearity of the full aerodynamic model delayed its completion, and as a result, most of the simulation results presented were obtained from the simplified 6DOF models. All simulations operate in continuous time using the Simulink® ode45 Dormand-Price differential equation solver.

The 6DOF models, THESIS1.MDL (Figure A.1) and THESIS1FILT (Figure A.14) employ the flat earth approximation (Equation 2.1) for their missile dynamics, and are streamlined models providing only the minimum number of subsystems required to quickly test guidance law operation.

All the air-to-air simulations use a point mass target simulation developed in [6]. The target dynamics are modeled with the following vector equation:[6]

$$\dot{x} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -\varpi & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & \varpi & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ v_x \\ y \\ v_y \\ z \\ v_z \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ a_z \end{bmatrix} \tag{2.13}$$

The target's lateral accelerations are modeled as a turn rate, ω, while the vertical acceleration is an input to the subsystem, a_z.

The TBMD model, THESISTBM.MDL (Figure A.23) uses the spherical earth model (Equation 2.2) for its missile dynamics. The target model used in this simulation involves a six dimensional state vector to simulate the dynamics of a point mass ballistic missile with no drag as shown below.

$$\dot{x} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ v_x \\ y \\ v_y \\ z \\ v_z \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} \tag{2.14}$$

The full aerodynamic model, THESIS3.MDL (Figure A.18) presented the greatest design challenge. In order to meet Stevens' and Lewis' requirement that the 6DOF equations be solved in the proper order, the flat earth dynamics block was completely redesigned (Figure A.21) The moments and forces on the missile are computed as outlined above, and then fed to the missile dynamics block as inputs. This should have resulted in a model that could be run in both open loop and closed loop operations.

Unfortunately, it was not possible to successfully close the loop with either the autopilot adapted from Blakelock, or any of several other autopilot designs.

It was possible to control the missile laterally, and for short periods vertically in an open loop by using the control deflection angles as inputs. It is likely that the failure of the closed loop operations was due to the inherent non-linearity of the model, and possibly the order in which Simulink® solves the computations in the various Matlab® function blocks. It may be possible to code both the aerodynamics and missile dynamics blocks as one inline Matlab® function to overcome this failure. This model is presented here as a point of departure for future research.

D. GUIDANCE LAWS

Five guidance laws were examined during this research, proportional navigation (PN), velocity compensated proportional navigation (VCPN), bang-bang, differential games (DG), and augmented proportional navigation (APN). The PN laws were used to establish baseline performance for comparison with the other guidance laws. The bang-bang and VCPN laws were examined as an extension of thesis work by Swee [11]. The DG and APN laws are derived in the optimal control literature and are the focus of using the kinematic boundary as measure of effectiveness [12], [13].

The geometry of a typical air-to-air missile engagement is shown in Figure 2.8. The object of the exercise is to steer the missile using only lateral accelerations in such a way that it hits the target. The steering commands should be optimal in some sense, minimizing miss distance at least, and possibly control effort (divert) or time to intercept.

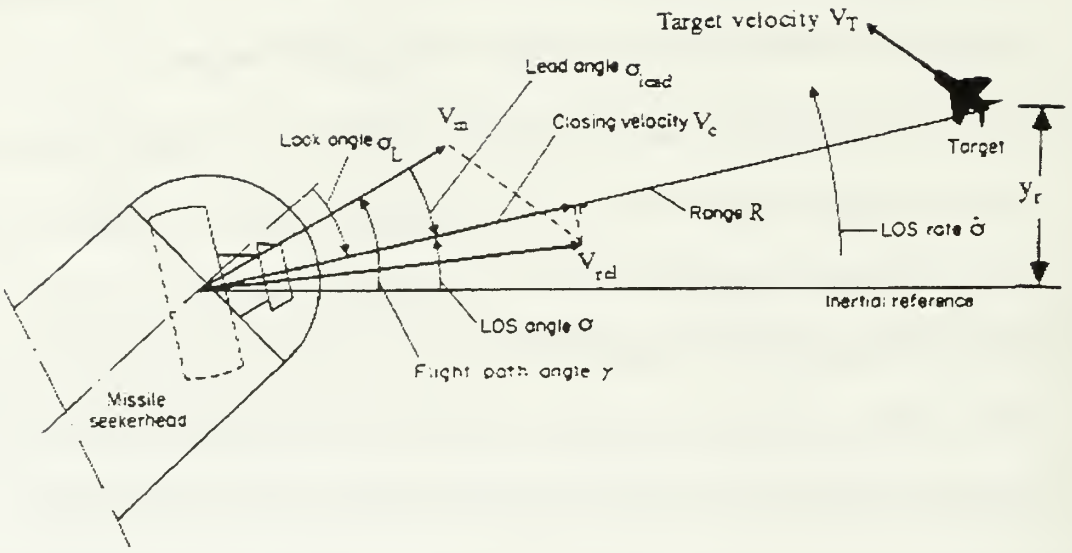


Figure 2.8. Typical missile engagement geometry. From [13].

1. Proportional Navigation

PN provides steering commands to the missile, which are proportional to the angular rate of the target's line of sight relative to a fixed reference. The command acceleration n_c is given by:[7]

$$n_c = \frac{N' V_C \dot{\sigma}}{\cos \sigma_L} \tag{2.15}$$

The cosine term in the denominator corrects the acceleration from the line of sight to the missile's y-axis.

PN with N'=3 has been shown to be optimal and guarantees a hit under the following conditions:[14]

- non-maneuvering target (no drag)
- missile speed greater than target speed
- target remains in missile's forward hemisphere

If the value of N' is sufficiently large, PN will always intercept a maneuvering target under these conditions. Blakelock implements PN as a turn rate, but his recommended values for a navigation constant equate to values of N' between three and five [8]. Higher values produce little improvement in performance. One other shortcoming of PN is that it does not account for the effect of the missile's dynamics (time constant) on the navigation solution.

2. Velocity Compensated Proportional Navigation

VCPN is an attempt to extend the basic PN law and account for the effect of drag on the missile. By adding a compensation term related to the missile's deceleration and the line of sight angle, the effect of the drag on the line of sight rate can be reduced. The VCPN law is given by:[13]

$$n_c = \frac{N' V_C \dot{\sigma}}{\cos \sigma_L} - \dot{V}_M \tan \sigma_L \tag{2.16}$$

In his thesis, Swee showed that if the range rate information VC is available to the missile, VCPN is no better than the basic PN law [11].

3. Bang-bang

Bang-bang guidance is a modification of PN in which the missile applies its full acceleration in the direction of the rate of change of the line of sight. The controls essentially "bang" on their stops whenever they are applied. This law would be useful in missiles that are controlled by thrusters that are not throttled and are either on or off. The bang-bang law is given by:[12]

$$n_c = A \frac{\operatorname{sgn}(V_C \dot{\sigma})}{\cos \sigma_L} \tag{2.17}$$

The bang-bang law use here is modified slightly because of the effects of drag. First, there is a dead band of 0.01 degrees per second in the line of sight rate before the guidance law takes effect, and second the acceleration at ranges greater than 5 km from the target is restricted to 5 g's. Inside 5 km, the acceleration is 30 g's. This was done to prevent the missile from expending all of its thrust overcoming the drag from 30 g turns immediately after launch.

4. Differential Games

Bryson and Ho develop a guidance law based on differential games theory in which the pursuer (missile) seeks to minimize a cost function based on the miss distance and the control effort while the evader (target) seeks to maximize the cost function and thus survive. Both players are assumed to have perfect knowledge of the other's state. Under these conditions, the evader's optimal strategy is to match the pursuer turn for turn as shown here [12].

$$a_e = \frac{c_e}{c_p} a_p \tag{2.18}$$

The pursuer's optimal strategy is more complicated, but after assuming the pursuer can turn at a faster rate than the evader, that minimum miss distance is infinitely more important than minimum control effort, and linearizing about a nominal collision course, the resulting control law is:[12]

$$n_c = \frac{3}{\left(1 - \frac{c_e}{c_p}\right)} V_C \dot{\sigma} \tag{2.19}$$

This is a variation of PN where the navigation constant can be varied statically at launch based on an estimate of the target's ability to maneuver, or dynamically with a

real-time estimate of the target's acceleration. Bryson and Ho say that c_p and c_e are constants related to the respective energies of the evader and pursuer, but closer analysis shows that they are also related to the ability to maneuver or available acceleration [12]. The control law implemented here uses a value of 30 g for c_p and estimates c_e from the output of the tracking filter.

5. Augmented Proportional Navigation

This guidance law is drawn from Lin, Reference [13], and is a simplification of an optimal guidance law that accounts for both target maneuver and missile dynamics. The APN law used here does not account for missile dynamics. It uses a twelve-dimensional state vector with the target's relative position, relative inertial velocity, target inertial accelerations, and missile body frame accelerations. The tracking filter estimates the first three, and the body frame accelerations are provided by the accelerometers. The guidance law is given by the following vector equation:[13]

$$\begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix} = \left(\frac{\Lambda}{t_{go}^2} \right) \begin{bmatrix} I_3 & t_{go} I_3 & \frac{t_{go}^2}{2} I_3 & \bar{0} \end{bmatrix} \begin{bmatrix} p_r \\ v_r \\ A_T \\ A_M \end{bmatrix} \tag{2.20}$$

The navigation constant Λ is computed for the full optimal guidance law as a function of t_{go} and the weighting functions on the miss distance and control effort. When the product of the weighting functions approaches zero, the navigation constant is equal to three. We have chosen a value of five to be consistent with the baseline PN law.

The position and velocity components of the state vector are relative to the missile and in inertial coordinates; therefore, they can be computed directly from the seeker

ranges and bearings. The time to go, t_{go}, is computed from the seeker range and range rate estimates.

Only the y- and z-components of the control are used. The x-component is ignored. Note that in this form, the missile accelerations are not used. A constant diagonal matrix is used in place of the "0" matrix to add the effect of the missile time constants in the full optimal guidance law.

E. TRACKING FILTER

The tracking filter is based on an alpha-beta-gamma filter design by Bar-Shalom and Li [15]. This is a constant gain filter and therefore it is less computationally intensive than an adaptive filter like the Kalman filter. Zarchan recommends the use of constant gain filters, in part because of computational load and also because of stability [7]. The DG and APN guidance laws require this tracking filter for their estimates of the target's acceleration

The filter is implemented as a MatLab® function ABGFILTER.M. It is interesting to note the use of the global variable XLAST to preserve the state estimate from time step to time step. The values for the filter gain were chose by trial and error from nomograms in Reference [15] to give the filter an initial settling time of less than two seconds as these simulations are initially noise free, and the choice of gains is dependent on the characteristics of the noise. The filter is a discrete time filter with a sampling frequency of 10 Hz. This was accomplished by placing the filter block between two zero order hold blocks. A sample of the filter's estimate of target acceleration with a 6 g turn three seconds prior to intercept is shown in Figure 2.9.

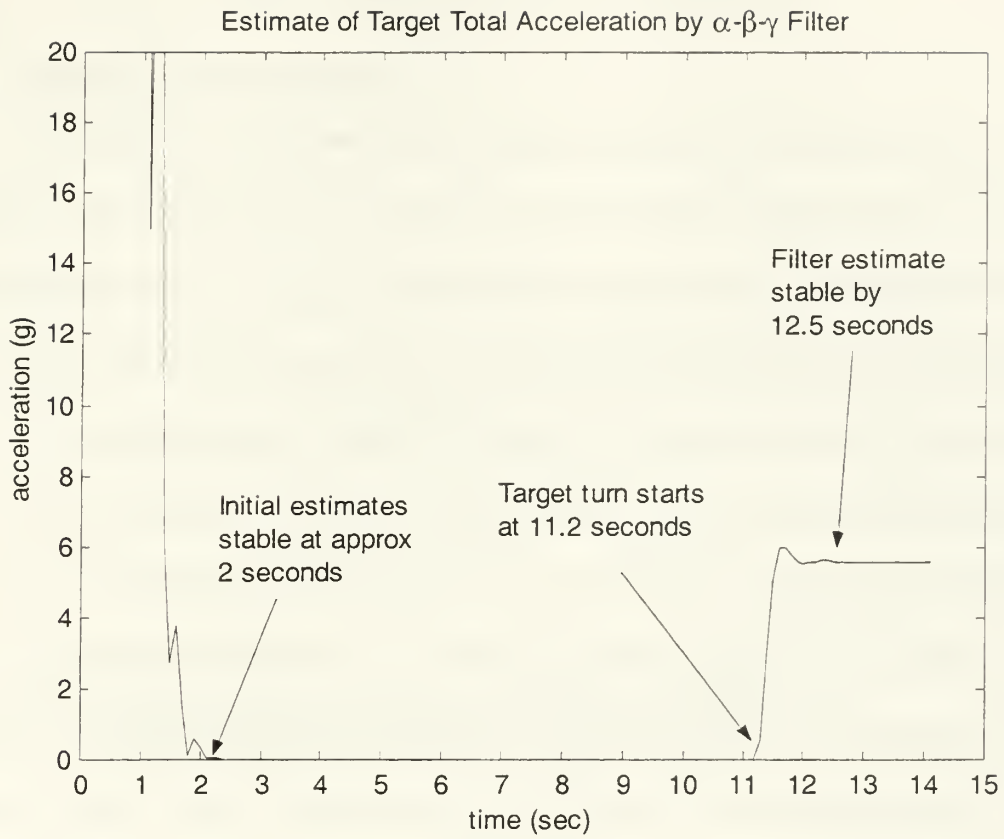


Figure 2.9. α–β–γ Filter Performance.

THIS PAGE INTENTIONALLY LEFT BLANK

III. GUIDANCE LAW TESTING

A. KINEMATIC BOUNDARY

Kinematics is the branch of mechanics dealing with pure motion without reference to the masses or forces involved. For the purposes of this research, a missile's kinematic boundary is the locus of points representing the maximum range at which a target may be successfully engaged as a function of relative bearing from the target at the start of the engagement given a noise-free guidance and control system. To the pilot, this is the "firing envelope," a critically important piece of information as it determines not only the success of an engagement, but the tactics required to prosecute the target. We chose the kinematic boundary as our measure of effectiveness for this reason. To the warfighter, graphs of average miss distance or control effort may be meaningful if he is a controls engineer, but a comparison of two guidance laws showing one to have a significantly larger firing envelope is far more useful. Figure 3.1 below shows a generic kinematic boundary (a circle) and is representative of the plots used in Chapter IV. The azimuth angles represent the relative bearing of the shooter from the target at the start of the engagement.

A successful engagement has a miss distance of less than 5 meters for these simulations. This figure is based on the warhead of the AMRAAM having a lethal radius of approximately 10 meters, and the size of a typical modern jet aircraft. Figure 3.2 shows the relationship of the 5 meter radius to a MiG-29 fighter. Clearly, a warhead exploding within 5 meters of the MiG-29 will do substantial if not fatal damage to the aircraft.

The Matlab® program files KBOUTER2.M and KBFILTER.M generate the kinematic boundaries. The resolution in range is 10 meters, and in azimuth is 5 degrees. These values were chosen as a compromise between speed of execution and plot detail. At these resolutions, a kinematic boundary can be generated in 9-12 hours with a Pentium® III, 700 MHz processor. One-degree resolution requires 48-60 hours, and 1 meter would take approximately 8-10 times longer.

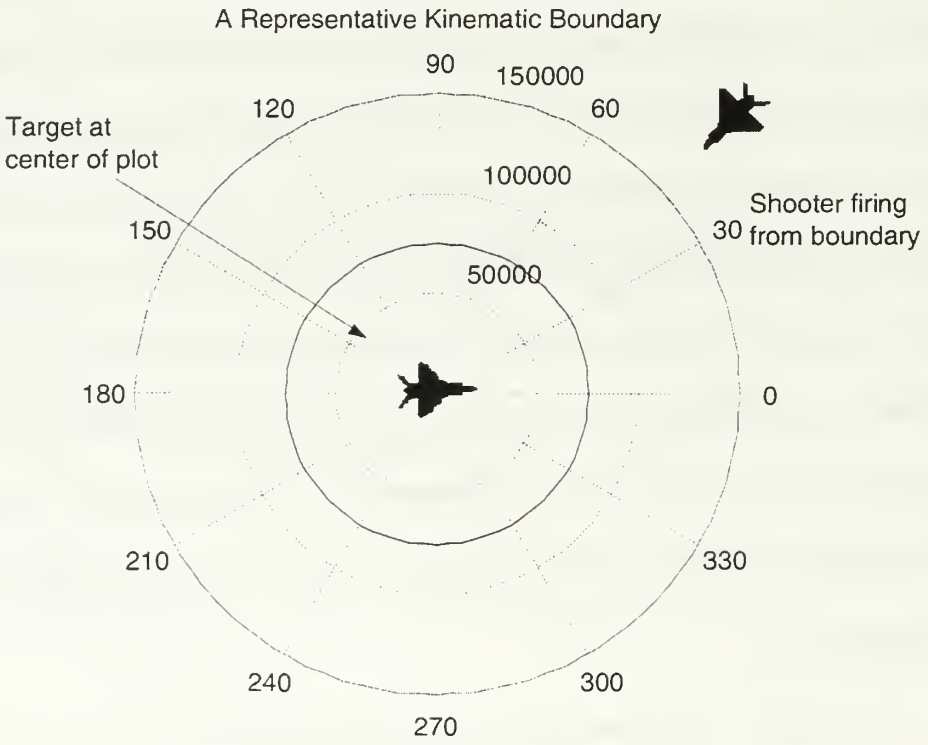


Figure 3.1. Kinematic boundary. The shooter is on the boundary pointing at the target at the start of the engagement.

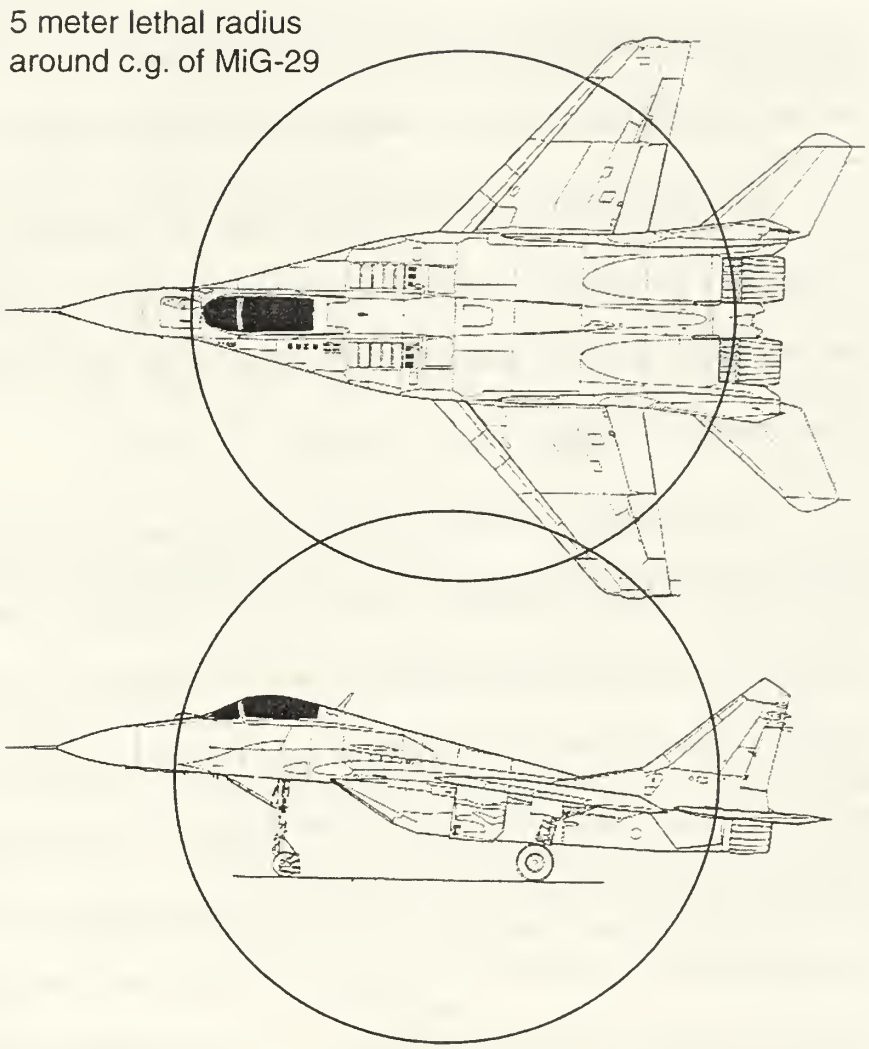


Figure 3.2. Comparison of a 5 meter warhead lethal radius to a MiG-29 aircraft. MiG-29 drawing is from [2].

B. TEST SCENARIOS

Candidate guidance laws are tested in three engagement scenarios:

- Non-maneuvering target, co-altitude at 6,000 meters
- Non-maneuvering cruise missile target at 50 meters
- Maneuvering target, co-altitude at 6,000 meters

The non-maneuvering target engagement is used as a baseline for comparison of performance. The engagement begins with target and shooter at 6,000 meters altitude, approximately 20,000 feet, and Mach 0.83. These values would be typical of an intruder making a high altitude ingress to a target, and a combat air patrol (CAP) on station.

The cruise missile engagement is intended to demonstrate the interceptor's ability to engage a low-altitude non-maneuvering target like the Tomahawk missile. The AMRAAM is launched from the CAP station at the target, which is at 50 meters, approximately 150 feet.

The maneuvering target engagement is the true test of missile performance. In this scenario, the target initiates a 6 g turn or "jink" toward the missile three seconds prior to impact. This turn toward the missile is most advantageous to the target as it forces the missile to make a tighter turn and expend more energy to keep up with the target than a turn away. The timing was chosen for two reasons. First, given that the missile will activate its terminal radar between 5-7 seconds prior to impact, and the time required for the target's sensors to detect the radar, alert the pilot, and have the pilot take evasive action, the aircraft would be established in its maneuver about three seconds prior to impact. Secondly, the missile's settling time is modeled to be 2.5 seconds, so a maneuver at three seconds puts increased stress on the guidance law to keep up with the maneuver.

C. CANDIDATE GUIDANCE LAWS

Table 3.1 shows how the guidance laws were tested.

Guidance Law	Non-maneuvering co-altitude	Non-maneuvering cruise missile	Maneuvering co-altitude
PN, N'=3	X		
PN, N'=5	X	X	X
PN, N'=7	X		X
VCPN with V_C			X
VCPN no V_C			X
Bang-bang	X		
DG	X		X
APN	X	X	X

Table 3.1. Guidance Law Test Plan

The VCPN and bang-bang laws were tested to confirm earlier work by Swee in his thesis [11].

D. NOISE STUDY

A study of the effect of seeker noise on missile performance was conducted using the PN (N'=5) guidance law. The study was run at the 135-degree azimuth test point with 100 realizations. The standard deviations of the noise signals were as follows:

- Range 50 meters
- Closing velocity 2 meter/second
- Bearing 1 degree
- Bearing rate .01 degree/second

THIS PAGE INTENTIONALLY LEFT BLANK

IV. COMPARISON AND ANALYSIS

A. PROPORTIONAL NAVIGATION LAWS

The PN guidance laws were tested to provide a baseline for comparison with the other guidance laws. Figure 4.1 shows the kinematic boundaries for the three PN laws against a non-maneuvering co-altitude target. Figure 4.2 shows the kinematic boundaries against the co-altitude, maneuvering target described above. Figures 4.3 and 4.4 are amplifications of the differences in performance of the three laws.

The N'=3 guidance law is the poorest performer of the three. While N'=3 has been shown to be optimal for a non-maneuvering target, the effect of drag on the missile is similar to a target maneuver along the line of sight. The discontinuities or "divots" in the N'=3 boundary are caused by drag slowing the missile more rapidly on those attack azimuths than others resulting in the missile slowing below the target's speed and stopping the simulation.

Clearly, the N'=5 law is an improvement for both scenarios. There is a slight improvement between N'=5 and N'=7 with a mean value of 315 meters for the non-maneuvering case, and 1,749 meters for the maneuvering case. The improvement from N'=3 to N'=5 has a mean value of 2,076 meters, non-maneuvering, and 7,555 meters maneuvering. Because of the relatively poor performance of the N'=3 law, N'=5 will be used as the comparison baseline for the other guidance laws.

Figure 4.5 is a comparison of the performance of the N'=5 law against a co-altitude target and against a cruise missile target at 50 meters altitude. The omni-directional reduction in range is due mainly to increased drag as the missile descends into the heavier air at lower altitudes.

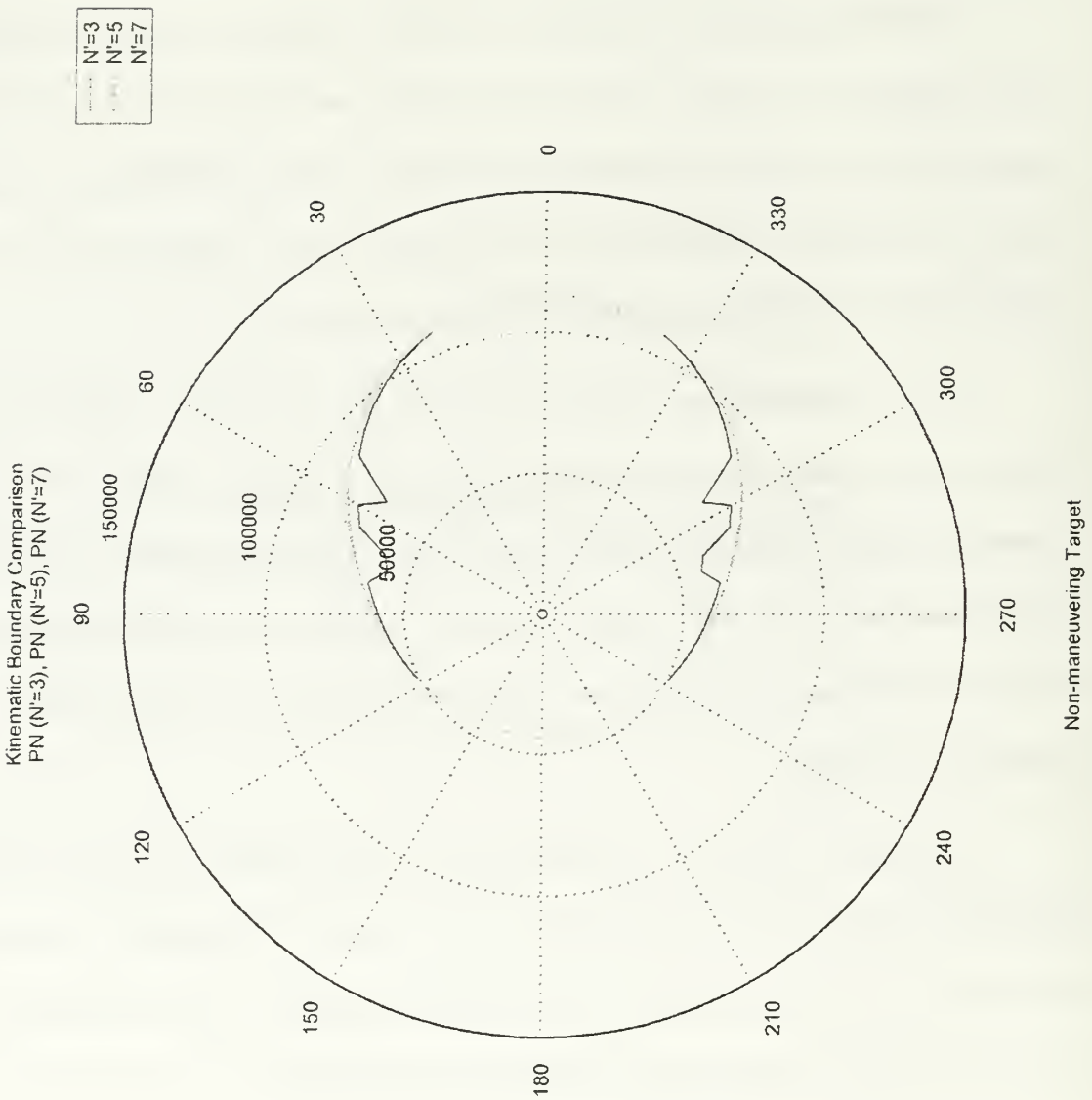


Figure 4.1. Kinematic boundary comparison of proportional navigation laws vs. non-maneuvering, co-altitude target at 6,000 meters and Mach 0.83.

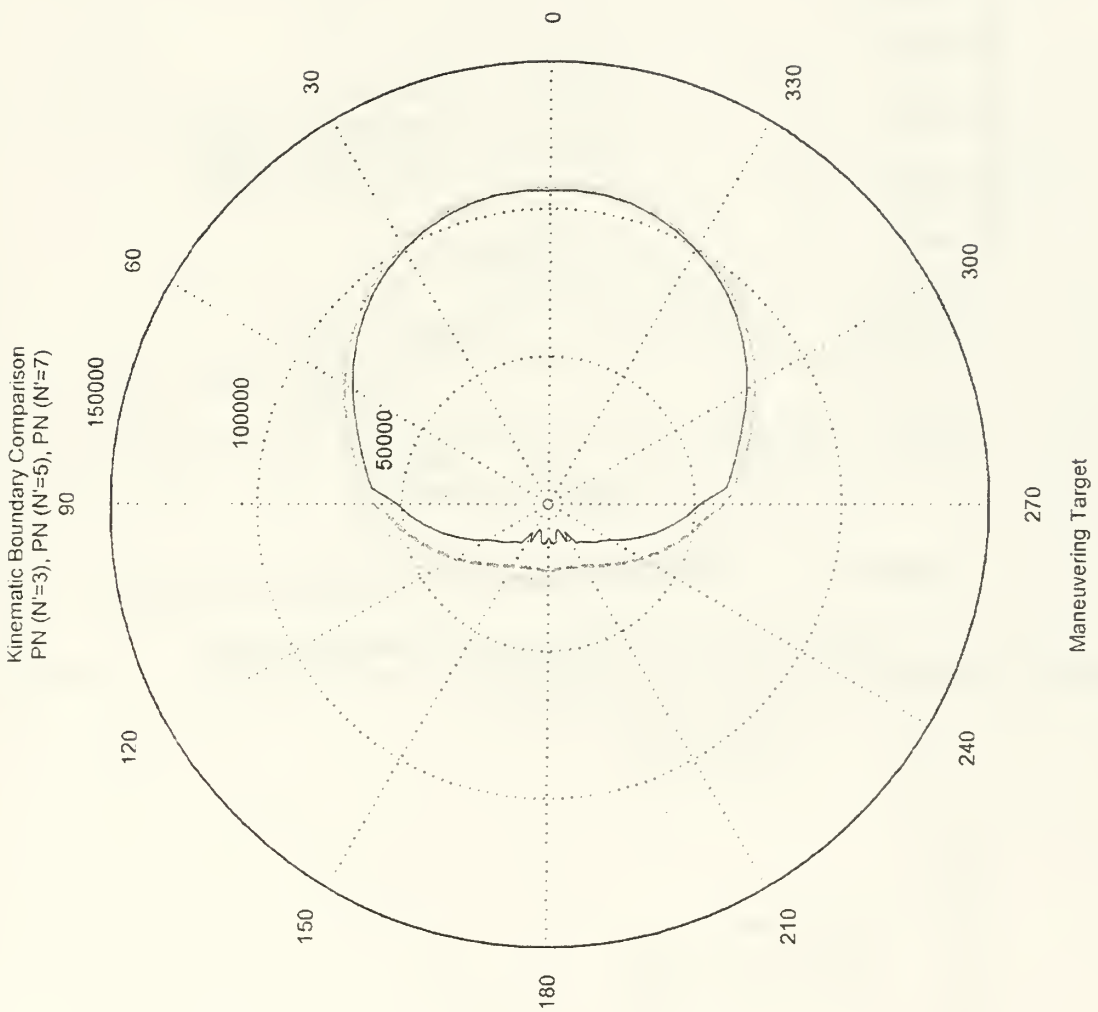


Figure 4.2. Kinematic boundary comparison of proportional navigation laws vs. maneuvering, co-altitude target at 6,000 meters and Mach 0.83. Target maneuver was a 6 g turn toward the missile at t_{go}=3 seconds.

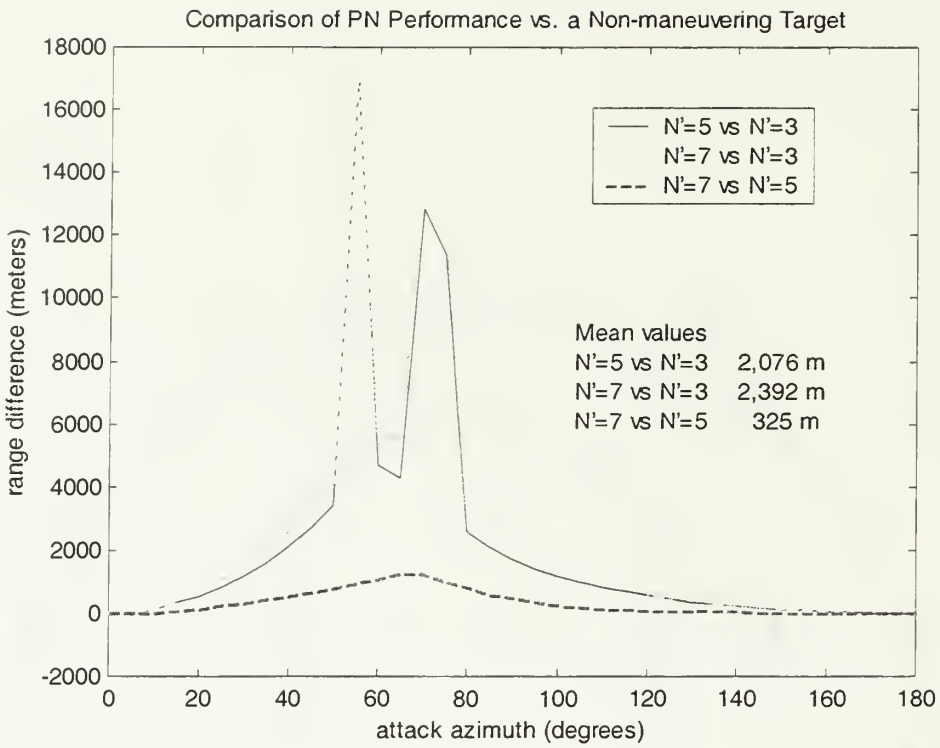

Comparison of PN Performance vs. a Non-maneuvering Target
range difference (meters)
attack azimuth (degrees)
N'=5 vs N'=3
N'=7 vs N'=3
N'=7 vs N'=5
Mean values
N'=5 vs N'=3 2,076 m
N'=7 vs N'=3 2,392 m
N'=7 vs N'=5 325 m

Figure 4.3. PN law performance vs. non-maneuvering target.

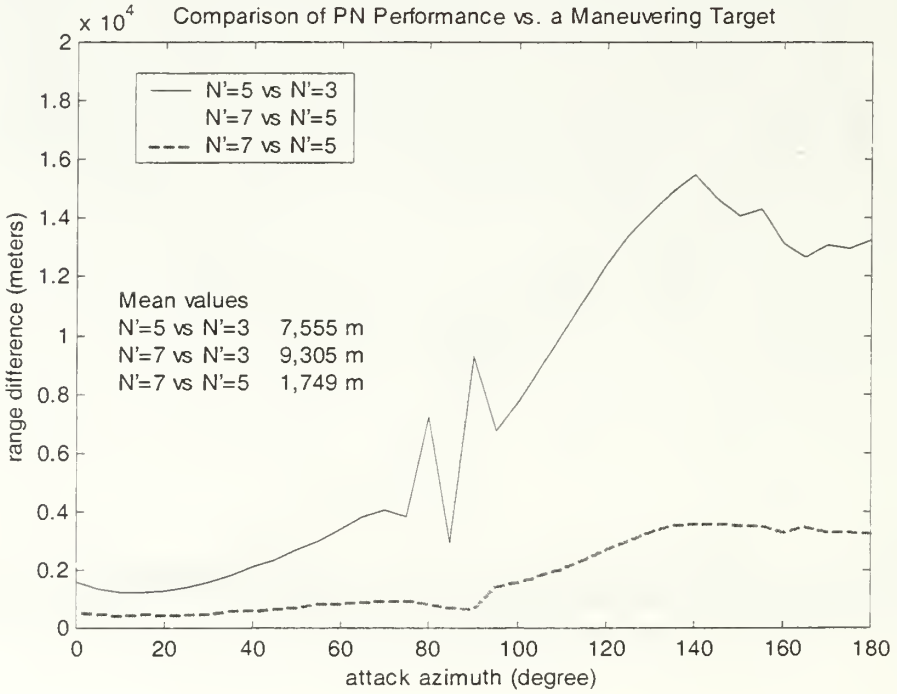


Figure 4.4. PN law performance vs. maneuvering target.

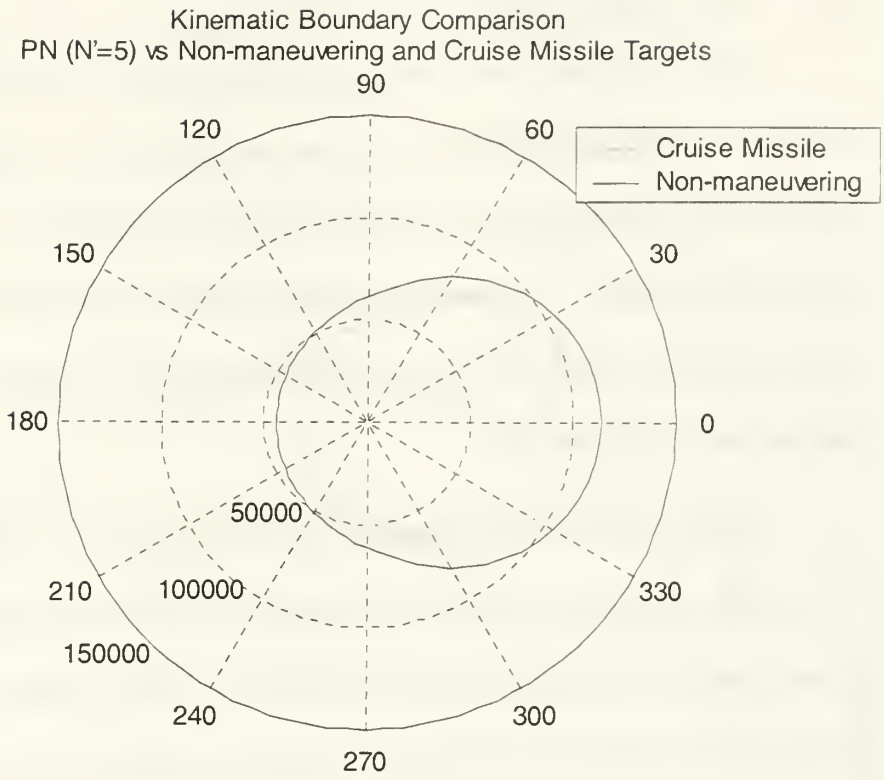


Figure 4.5. Kinematic boundary comparison of PN vs. non-maneuvering and cruise missile targets.

B. VELOCITY COMPENSATED PN LAWS

The VCPN laws were tested as a continuation of Swee's thesis research [11]. A PN law with a fixed navigation constant, angles only, no V_C information, was tested against the non-maneuvering, co-altitude target. This simulates a guidance law like that used by Sidewinder. The gain was computed with N'=5 and a fixed closing velocity of 750 meters per second. VCPN laws with and without V_C information are compared to this law, and to PN with N'=5.

Figure 4.6 shows the kinematic boundaries for each of these laws. The VCPN law without V_C information is clearly an improvement over the angles only PN law, while the addition of V_C information to the VCPN law actually reduces the range. Since the incorporation of V_C information also includes the deceleration of the missile along the line of sight, the velocity compensation term adds nothing to the guidance law's performance. Neither of the VCPN laws performed as well as the full PN law.

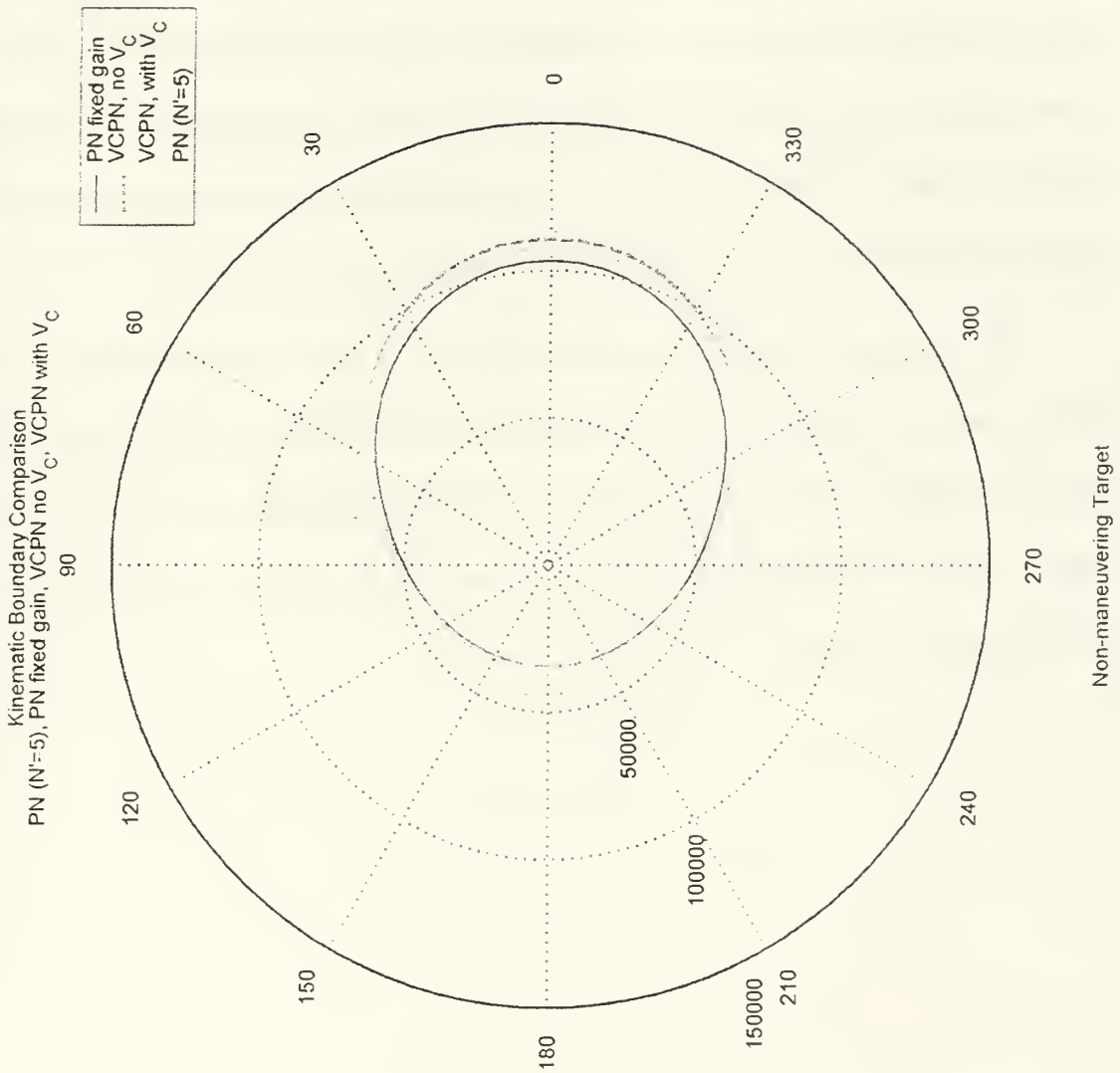


Figure 4.6. Kinematic boundary comparison of VCPN laws vs. a non-maneuvering, co-altitude target at 6,000 meters and Mach 0.83.

C. BANG-BANG

The bang-bang guidance law was also tested as a continuation of Swee's work.[11] The bang-bang law is used throughout the engagement to determine the effect of drag on its performance. As seen in Figure 4.7, bang-bang is clearly outperformed by the baseline PN law. Because of the aerodynamic drag on the missile, the guidance law must expend more energy in the end game when the line of sight angular rates begin to increase. There is a synergistic effect: as the angular rate increases, the missile must turn harder, generating more drag, which causes the angular rate to increase.

Note that the bang-bang law's performance is highly aspect dependent. The enhancement in the target's forward hemisphere is most noticeable. The effect could be useful in TBMD work where the goal is to place the interceptor ahead of the target. Further, for an exo-atmospheric interception, the effect of drag on the bang-bang law would be greatly reduced.

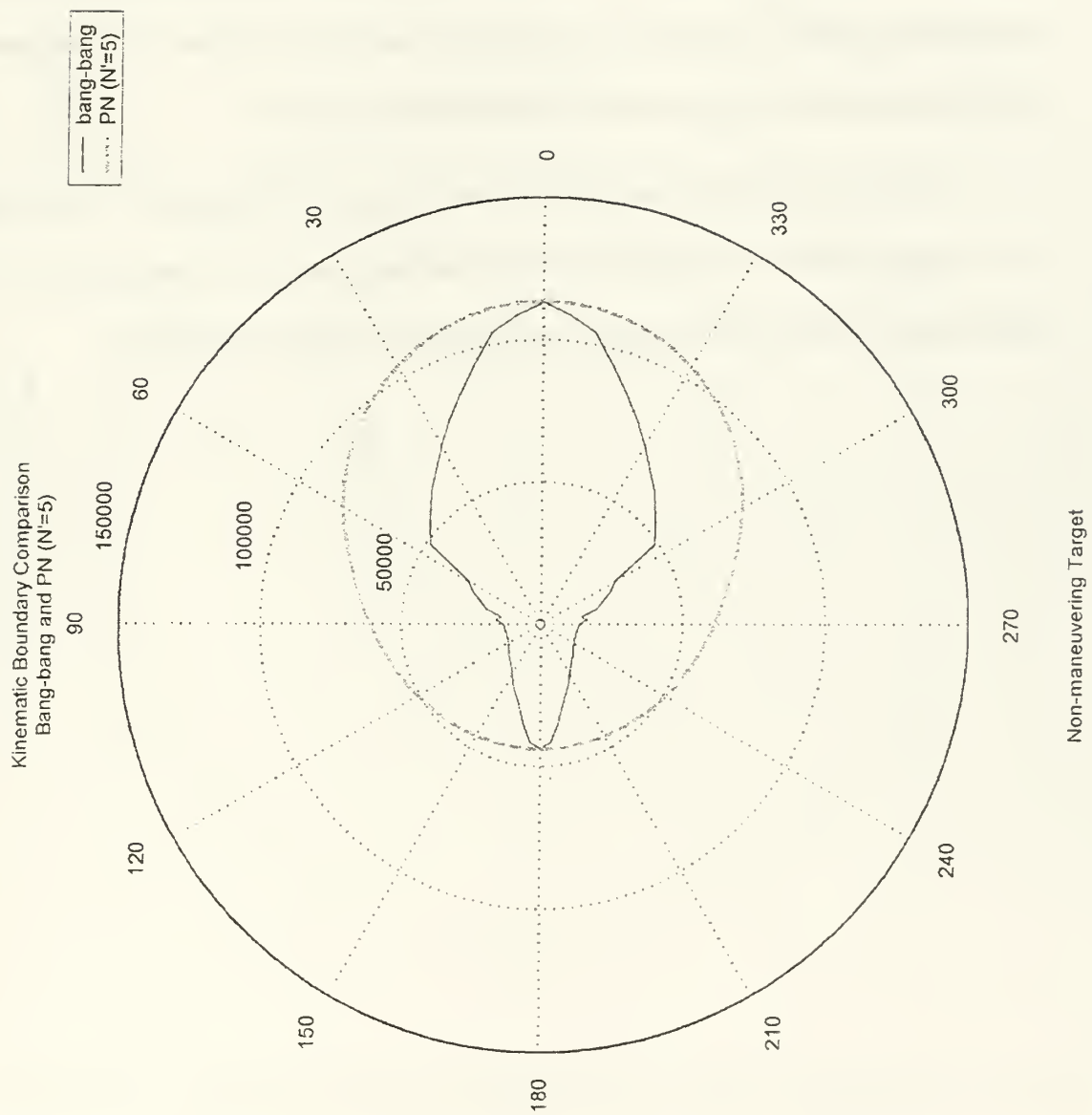


Figure 4.7. Kinematic boundary comparison of the bang-bang law vs. a non-maneuvering, co-altitude target at 6,000 meters and Mach 0.83.

D. DIFFERENTIAL GAMES

The differential games law was tested against both the non-maneuvering and maneuvering co-altitude targets. Figure 4.8 shows its performance against the non-maneuvering target, and Figure 4.9 against the maneuvering target. In both cases the performance showed no improvement over the baseline PN law.

This law is a modification of PN with scheduling of the navigation constant based on the tracking filter's estimate of the target's total acceleration. It is clear that gain scheduling is not sufficient to increase the kinematic boundary of the PN law.

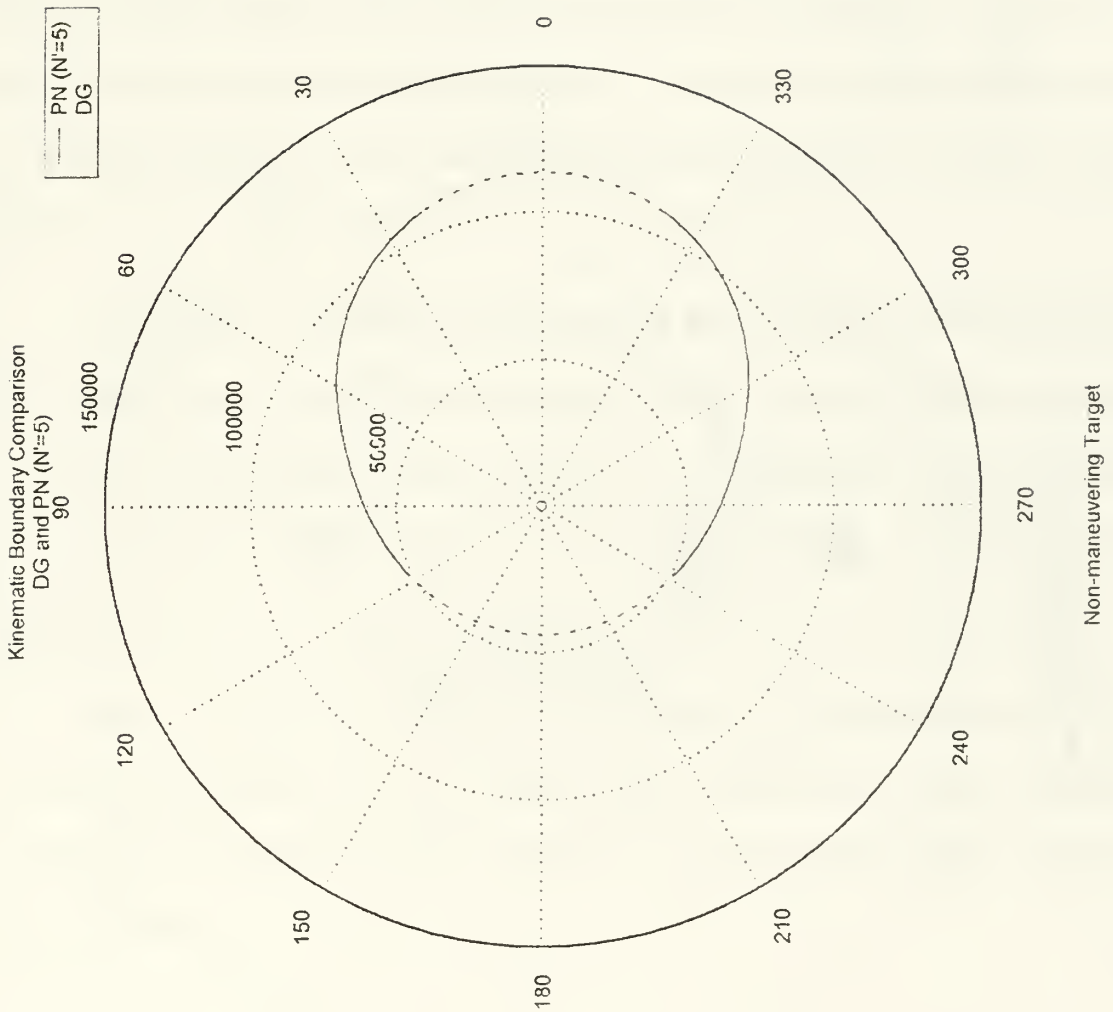


Figure 4.8. Kinematic boundary comparison of the differential games law vs. a non-maneuvering, co-altitude target at 6,000 meters and Mach 0.83.

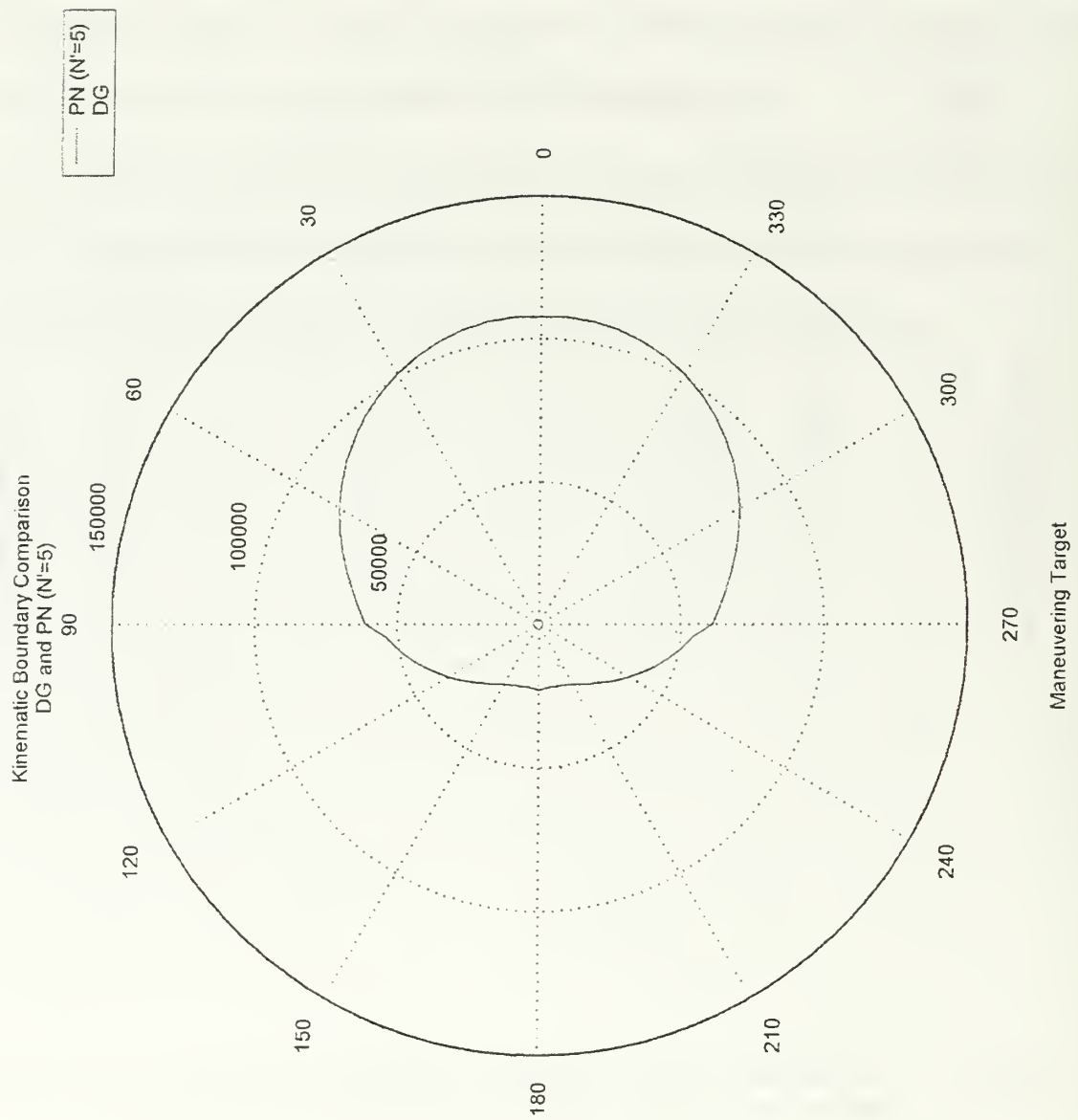


Figure 4.9. Kinematic boundary comparison of the differential games law vs. maneuvering, co-altitude target at 6,000 meters and Mach 0.83. Target maneuver was a 6 g turn toward the missile at t_{go}=3 seconds.

E. AUGMENTED PROPORTIONAL NAVIGATION

The APN law was tested against both the non-maneuvering and maneuvering, co-altitude targets. Against the non-maneuvering target, the APN law's performance is identical to the baseline PN law except for a small "divot" at 100 degrees. Figures 4.10 shows the results against the maneuvering target. The jagged boundary is an artifact of the azimuthal resolution, and is smoothed out when the resolution is reduced to one degree.

The APN law is clearly better in the target's rear hemisphere and forward of 60 degrees relative to the nose. In the forward quarter from 90 degrees to 60 degrees there is a reduction in performance compared to the PN law. The mean improvement in the APN law is 4.45 km for all aspects, 1.24 km in the forward 120 degrees, and 8.48 km in the rear hemisphere.

Figure 4.11 shows the results of APN against the cruise missile target. The kinematic boundary for the APN law is clearly smaller than PN law. Azimuth resolution was reduced to 10 degrees for this comparison to keep the APN simulation under 48 hours in real time.

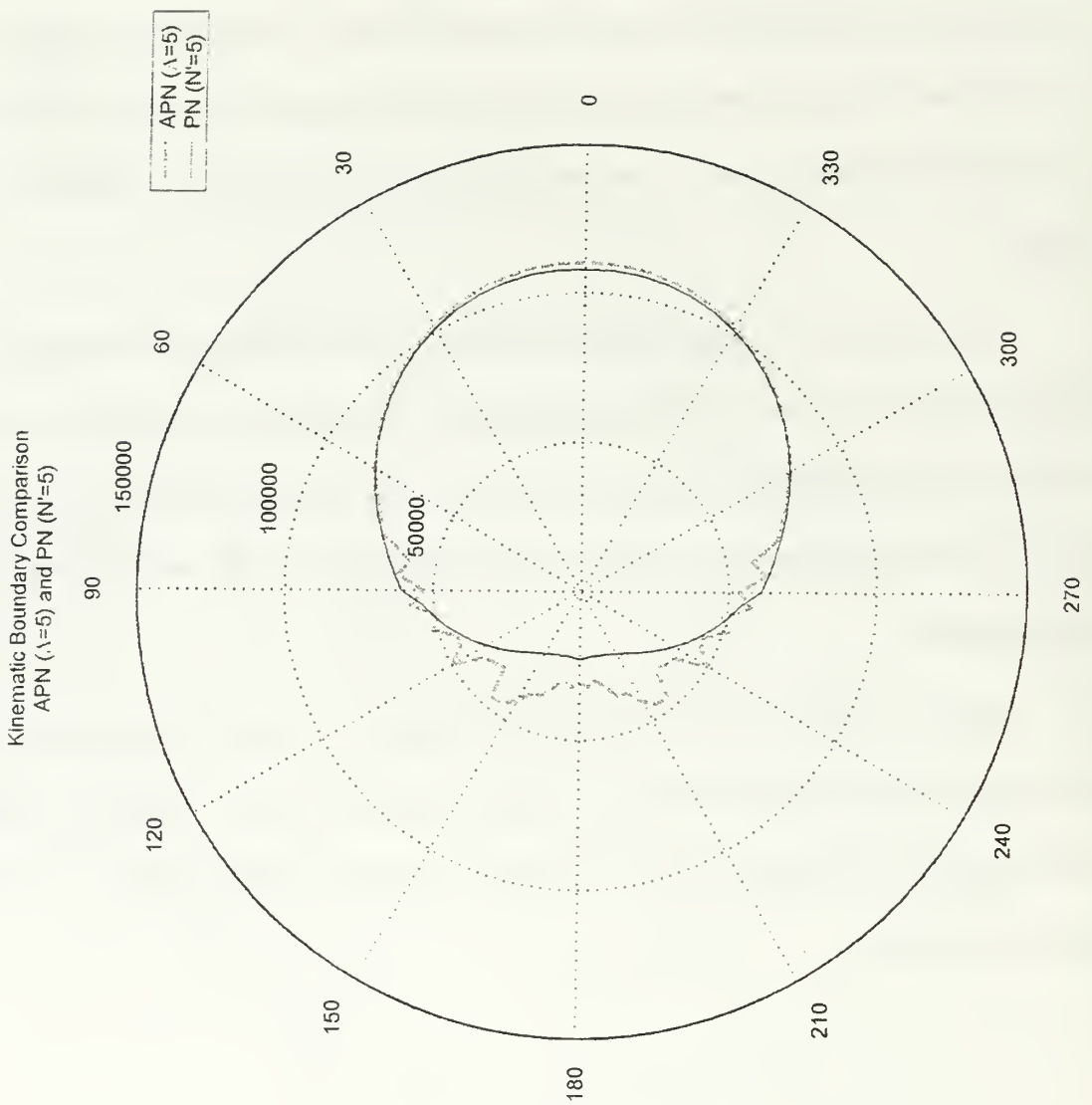


Figure 4.10. Kinematic boundary comparison of APN vs. maneuvering, co-altitude target at 6,000 meters and Mach 0.83. Target maneuver was a 6 g turn toward the missile at t_{go}=3 seconds.

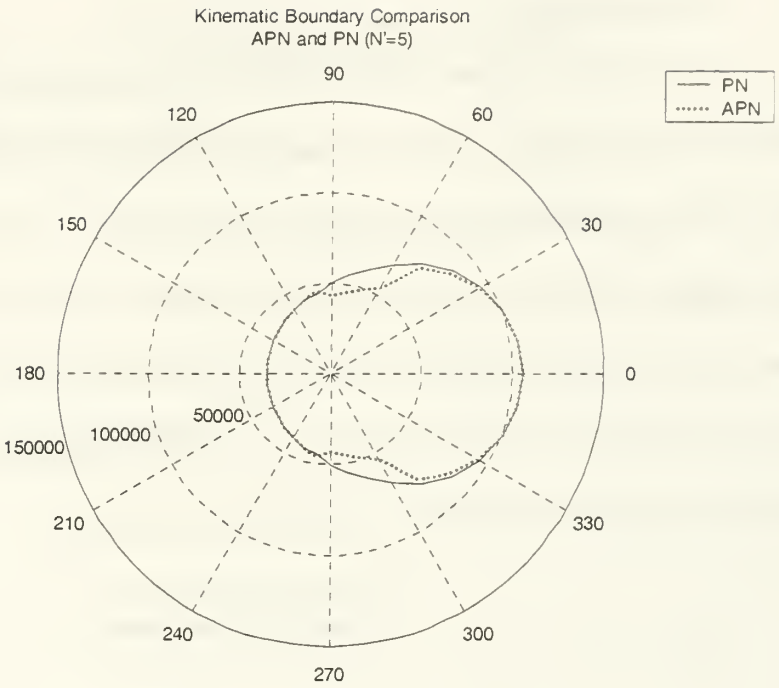


Figure 4.11. Kinematic boundary of APN and PN vs. cruise missile target. Azimuth resolution is 10 degrees.

F. NOISE STUDY

A study of the effects of noise on missile performance was conducted at the 135-degree azimuth test point. Using the range for the kinematic boundary of the baseline PN law, it was not possible to hit the target in 100 realizations. The test point was moved in approximately 2 km to 45,500 meters and another 100 realizations were generated. Figure 4.12 is a scatter plot of the x and y miss distances for the 100 realizations. Figure 4.13 is the distribution of the Euclidean miss distances. 92 percent of the samples were within the required miss distance of 5 meters to be called hits.

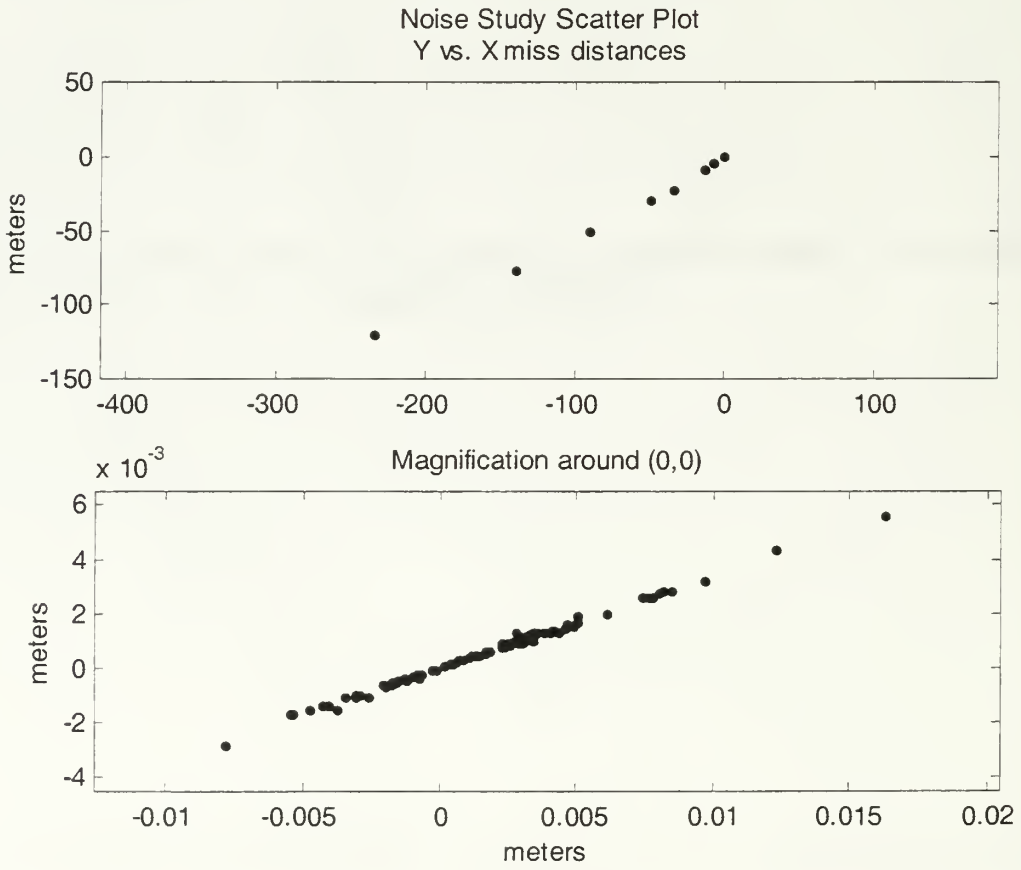


Figure 4.12. Scatter plot of x and y miss distances for a noisy seeker.

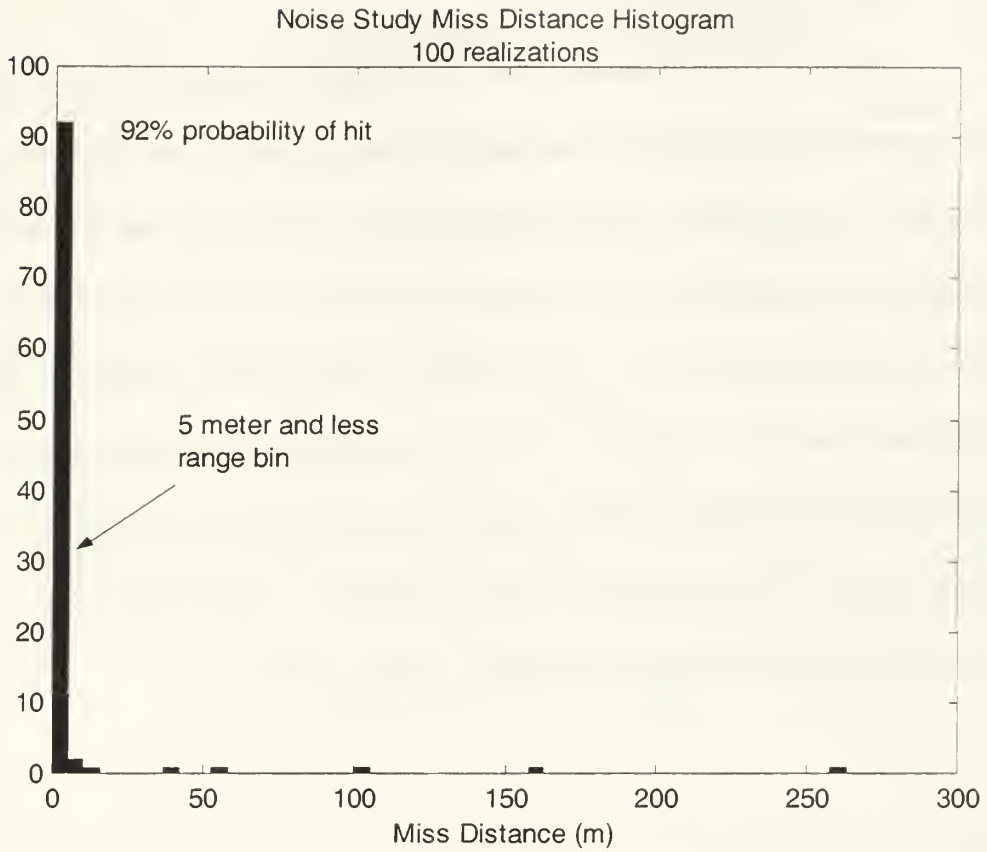


Figure 4.13. Histogram of missile miss distances with a noisy seeker. 100 realizations. Probability of hit is 92%.

G. TBMD DEMONSTRATION

Figure 4.14 is a demonstration of the 6DOF model's ability to simulate a TBM interceptor. The target missile was launched from the equator on a northeasterly heading. The range of this missile is approximately 400 km. The interceptor was launched from a position 150 km north of the target launch site. The target's initial velocity vector, [v_x v_y v_z] in ECI coordinates, was [1200 10 1000]. The velocity profiles for the target and interceptor are shown in Figure 4.15. The interceptor was steered toward the target's apogee for the first 30 seconds of flight, and then followed the baseline PN law to an interception 2.2 meters ahead of the target. The plot is in ECI coordinates, the surface of the earth is approximately the bottom grid. North is to the right.

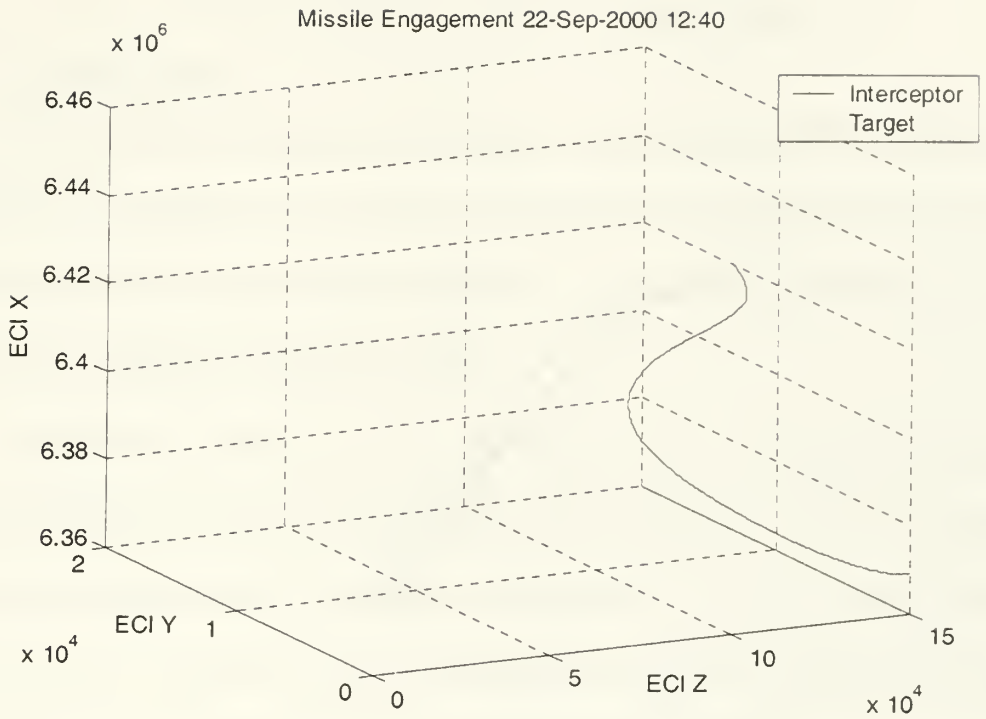


Figure 4.14. TBMD engagement by RIM-67 SM-2 (ER). Miss distance at intercept was 2.2 meters.

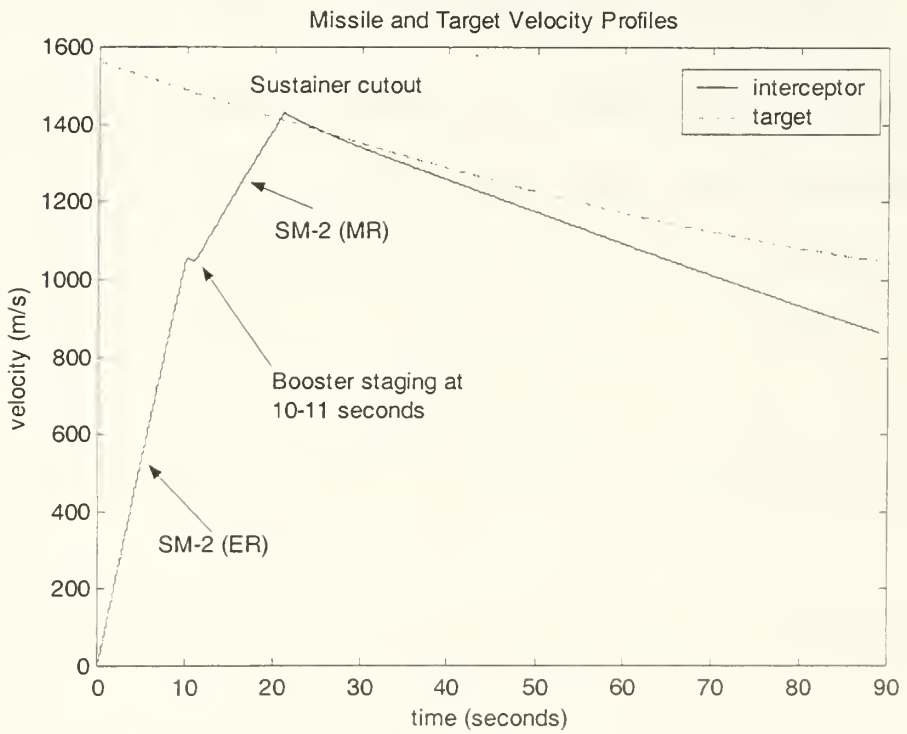


Figure 4.15. Interceptor and target velocity profiles for TBM demonstration.

THIS PAGE INTENTIONALLY LEFT BLANK

V. CONCLUSIONS AND FUTURE RESEARCH

A. CONCLUSIONS

The kinematic boundary is a natural, intuitive method of comparing the performance of guidance laws. Its form is immediately recognizable to the warfighter, and provides exactly the information required to make an informed decision as to which guidance laws would be of operational value.

The VCPN laws showed the expected improvement over the fixed gain PN law, and the VCPN law with range rate information did not perform as well as the VCPN law without range rate information. Neither law performed as well as the baseline PN law. The bang-bang law showed an unusual kinematic boundary with ranges in the target's forward hemisphere greatly extended over the rear hemisphere. The aspect dependence of this law and approximately 50 percent reduction in range throughout most of the envelope make this law a poor choice for guidance from launch to intercept.

Under the conditions simulated here, an optimal control law, the augmented proportional navigation law, will perform better than a proportional navigation law throughout most of the kinematic boundary. Overall, the APN law's kinematic boundary was 4.45 km better than the PN law, on average. In the forward 120 degrees, the average improvement was 1.24 km and in the rear hemisphere, it was 8.48 km. This represents a 1.4 percent improvement over the PN law for head-on engagements, and a 25 percent improvement for rear hemisphere engagements.

The 6DOF model has been demonstrated as a test platform for evaluating guidance laws for use in the TBMD arena.

B. FUTURE RESEARCH

This work suggests several lines of future research. First, the guidance laws tested here should be tested with a noisy seeker to determine the effect of noise on their performance. Second, the full aerodynamic model designed here needs to be taken to the point where it is fully operational. Third, the TBM simulation could be used to study the effects of guidance law selection on the Navy's Linebacker TBMD capability. Fourth, the models could be used for a comparison of the kinematic boundaries of other missiles systems, particularly those that are potentially hostile to look for possible tactical advantages. Finally, the models could be used to test new guidance laws that will be developed future researchers.

APPENDIX A. SIMULINK® MODELS

The block diagrams in this appendix represent the four models used in this research. Sub-blocks which are not changed from the simplified 6DOF model in later models are not included with those models. The four models begin on the following pages:

- Simplified 6DOF 64
- 6DOF with tracking filter 77
- Full aerodynamic model 81
- TBMD interceptor model 86

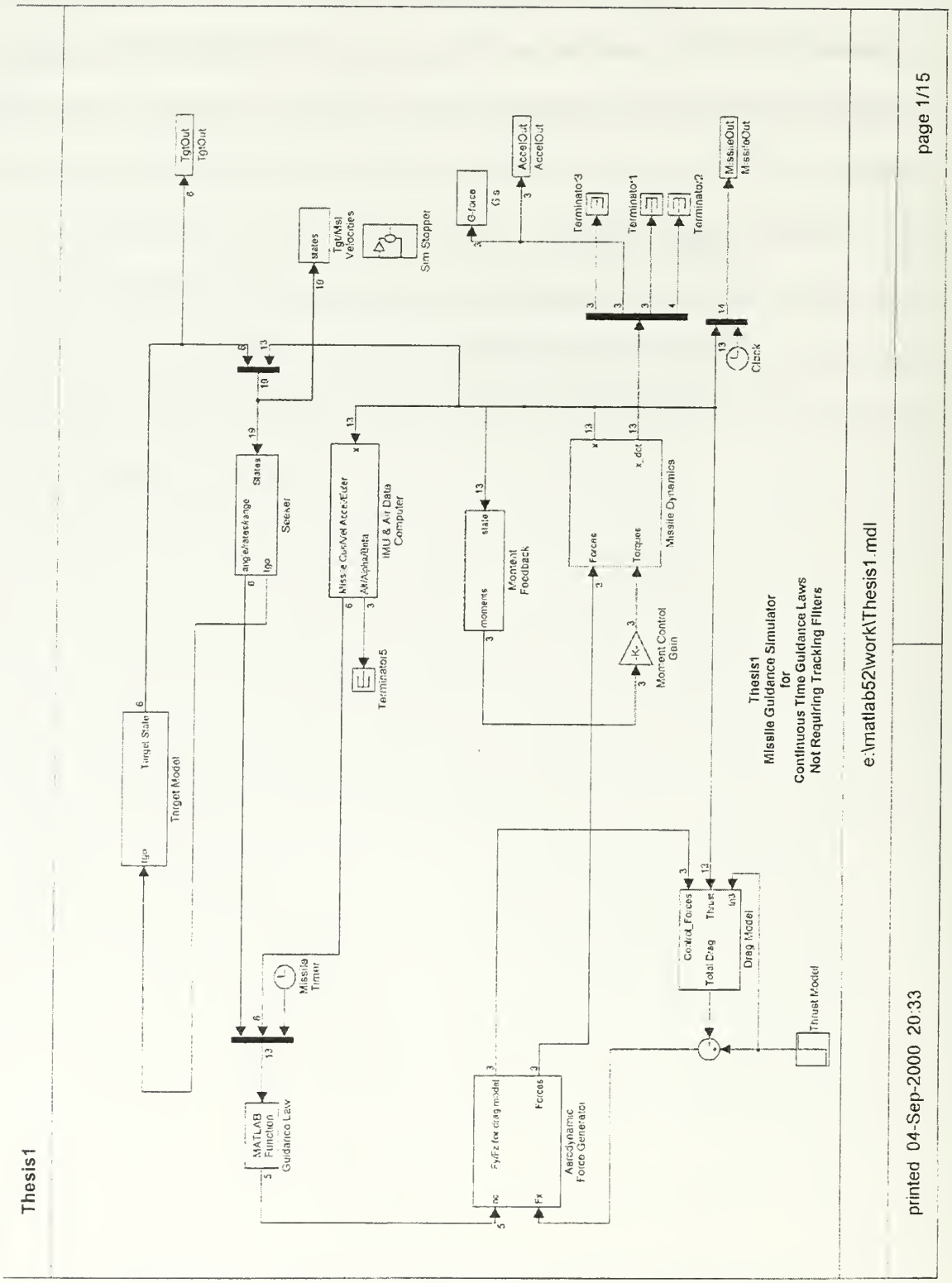


Figure A.1. Simplified 6DOF model without tracking filter.

Figure A.2. Simplified 6DOF Aerodynamic Force Generator.

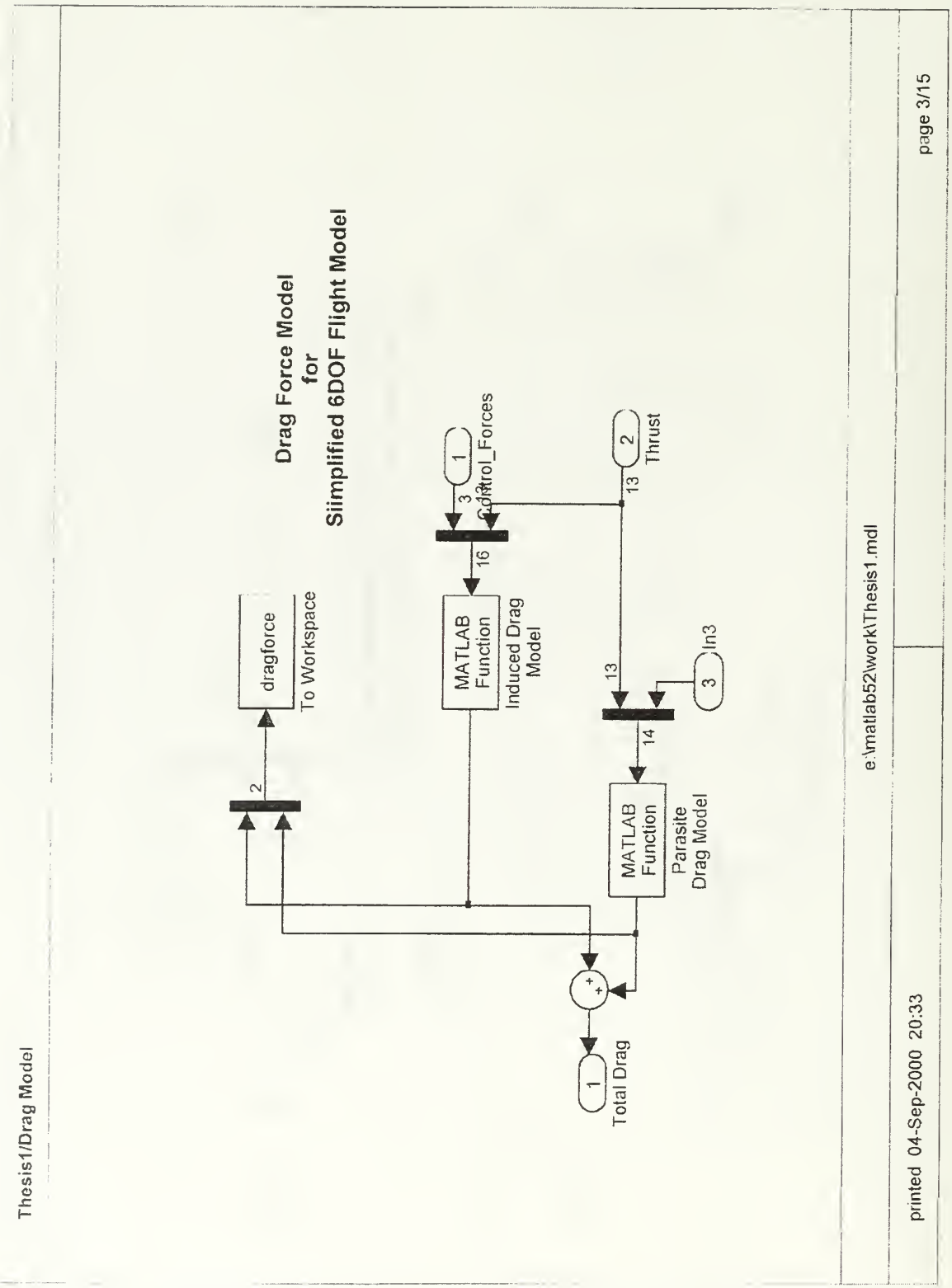


Figure A.3. Simplified 6DOF Drag Model. Function blocks are DRAGINDUCED.M and DRAGTHESIS.M.

Figure A.4. Flat Earth 6DOF missile dynamics.

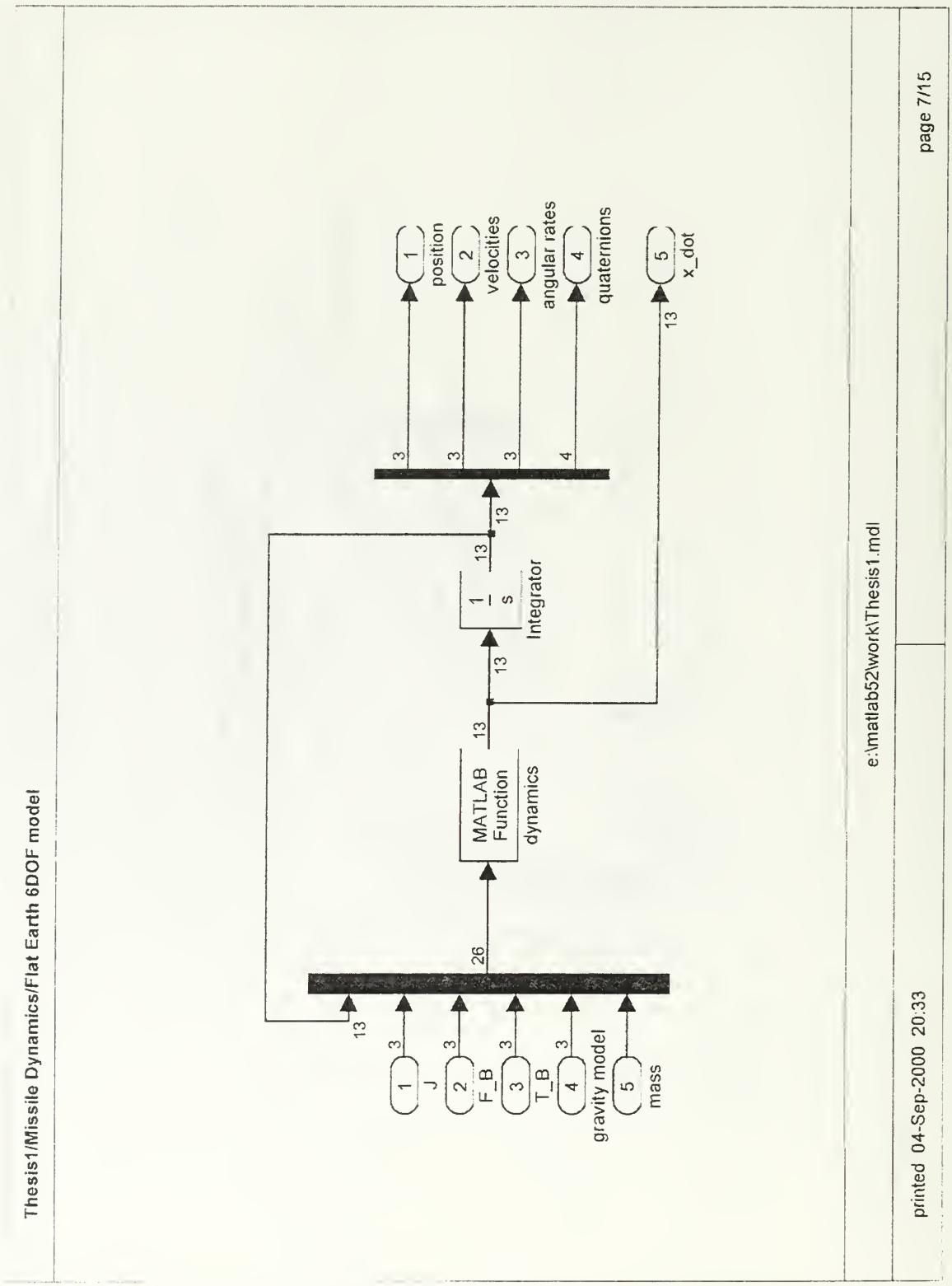


Figure A.5. Internal missile dynamic model. SIXDOFDYN.M is the function block.

Figure A.6. Aerodynamic moment feedback.

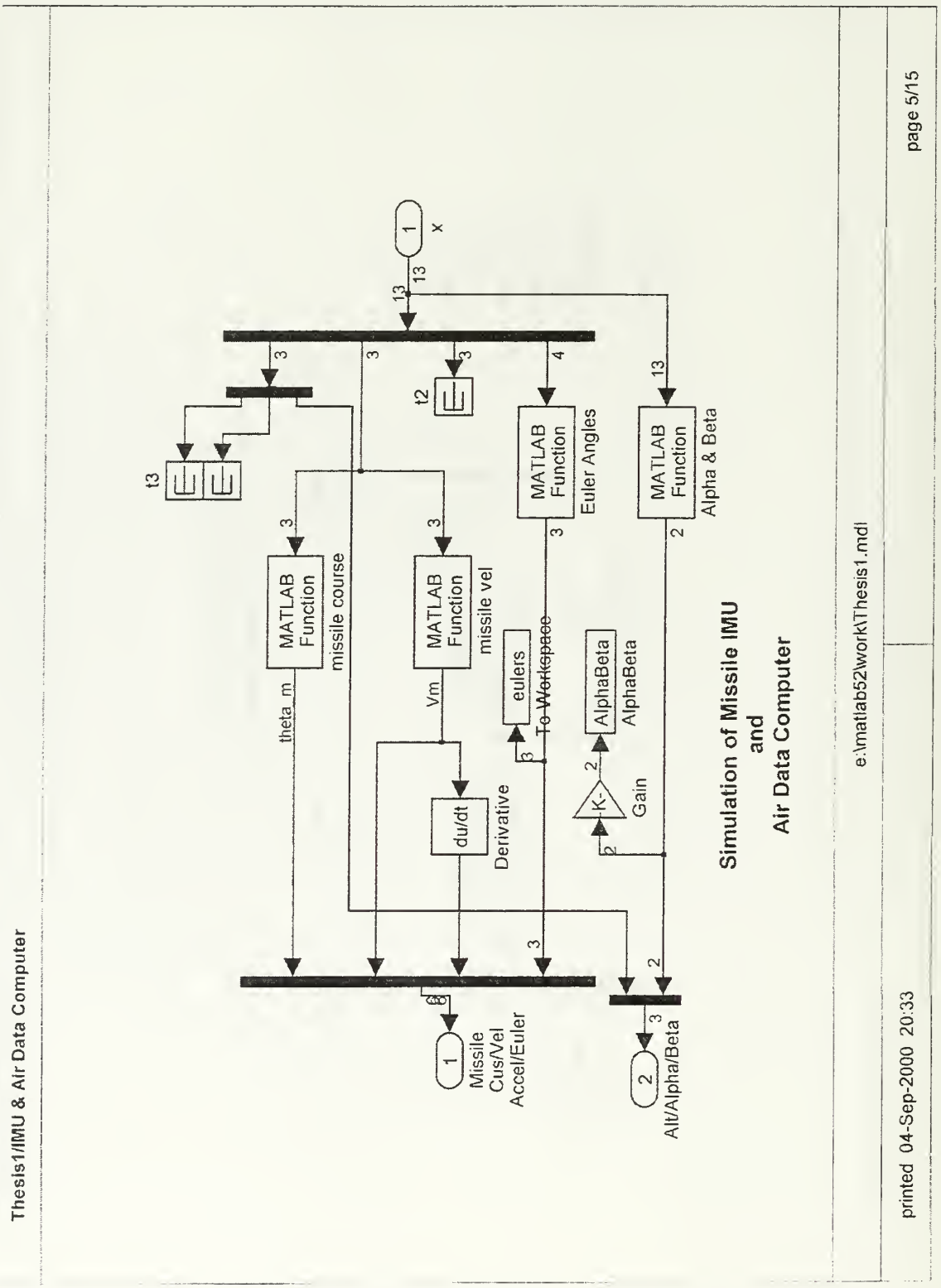


Figure A.7. Missile IMU and air data computer. Function blocks are Q2EULER.M and ALPHABETA.M.

Figure A.8. Missile seeker model.

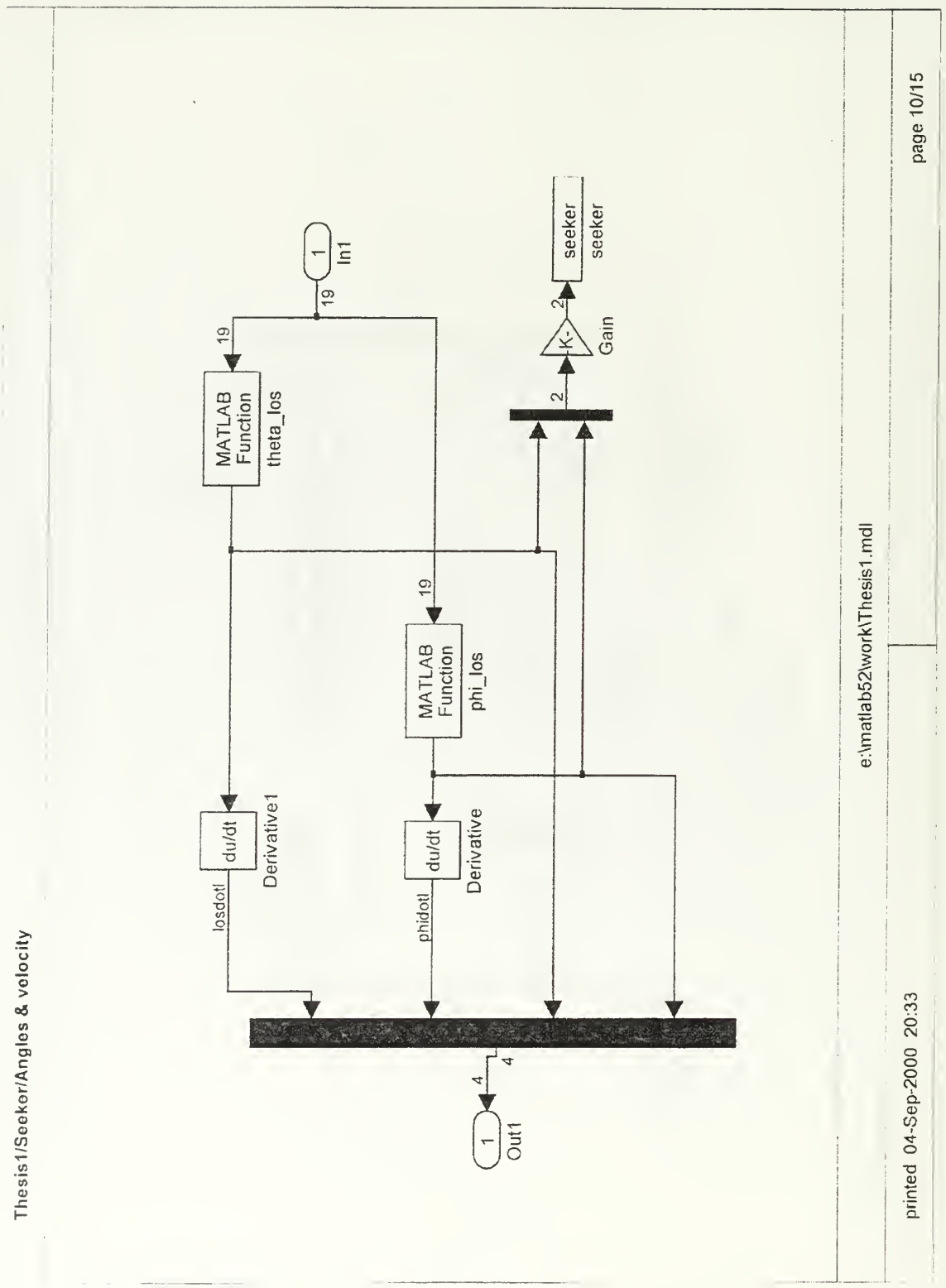


Figure A.9. Gimbal angles and rates.

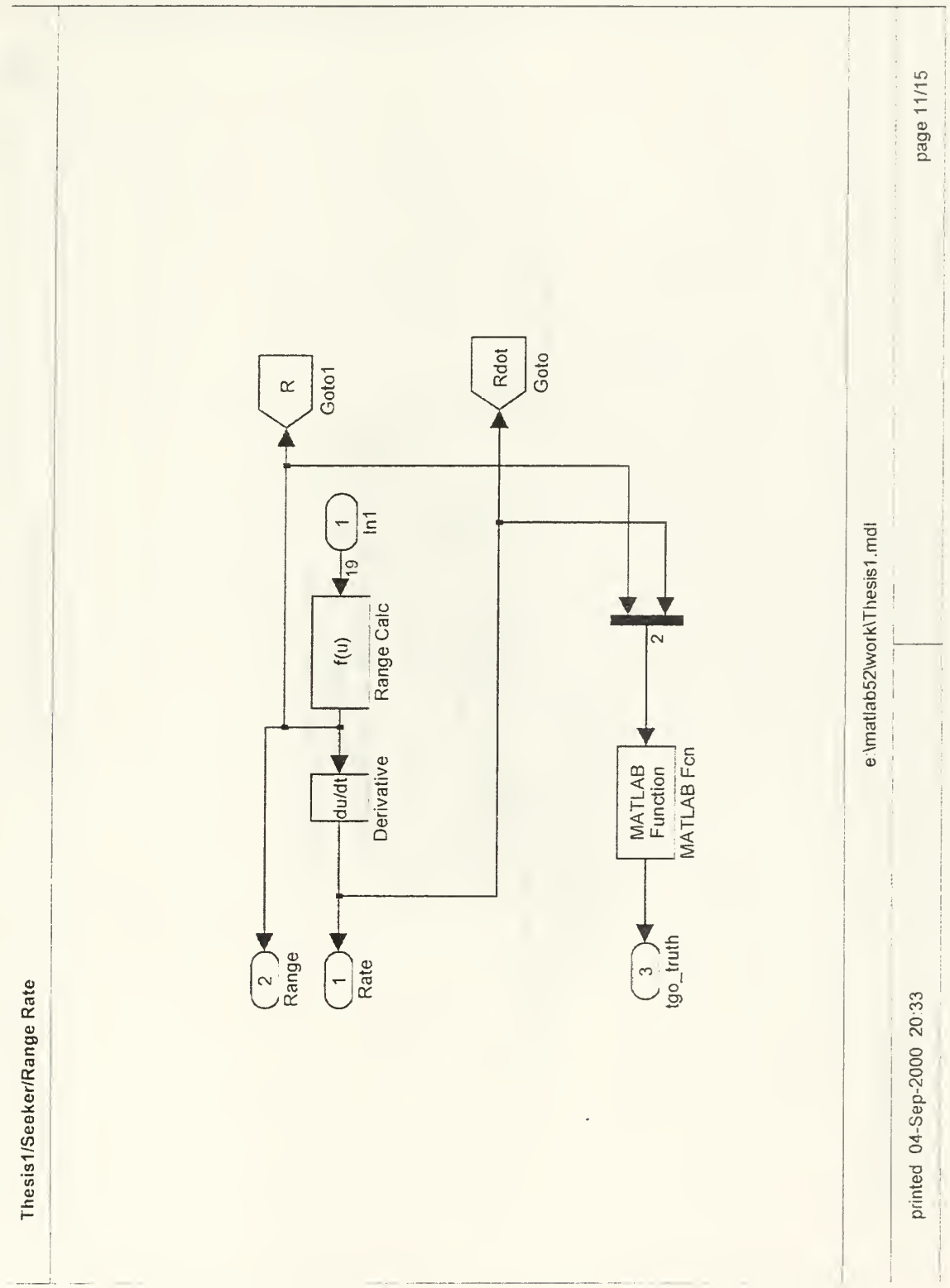


Figure A.10. Range, range rate, and time-to-go. Function block is TGO.M.

Figure A.11. Target dynamics model. Function block is DYNAMIC3D.M.

Figure A.12. Target turn generator. Switch threshold is set to 3 seconds.

Figure A.13. Target and missile velocity computation.

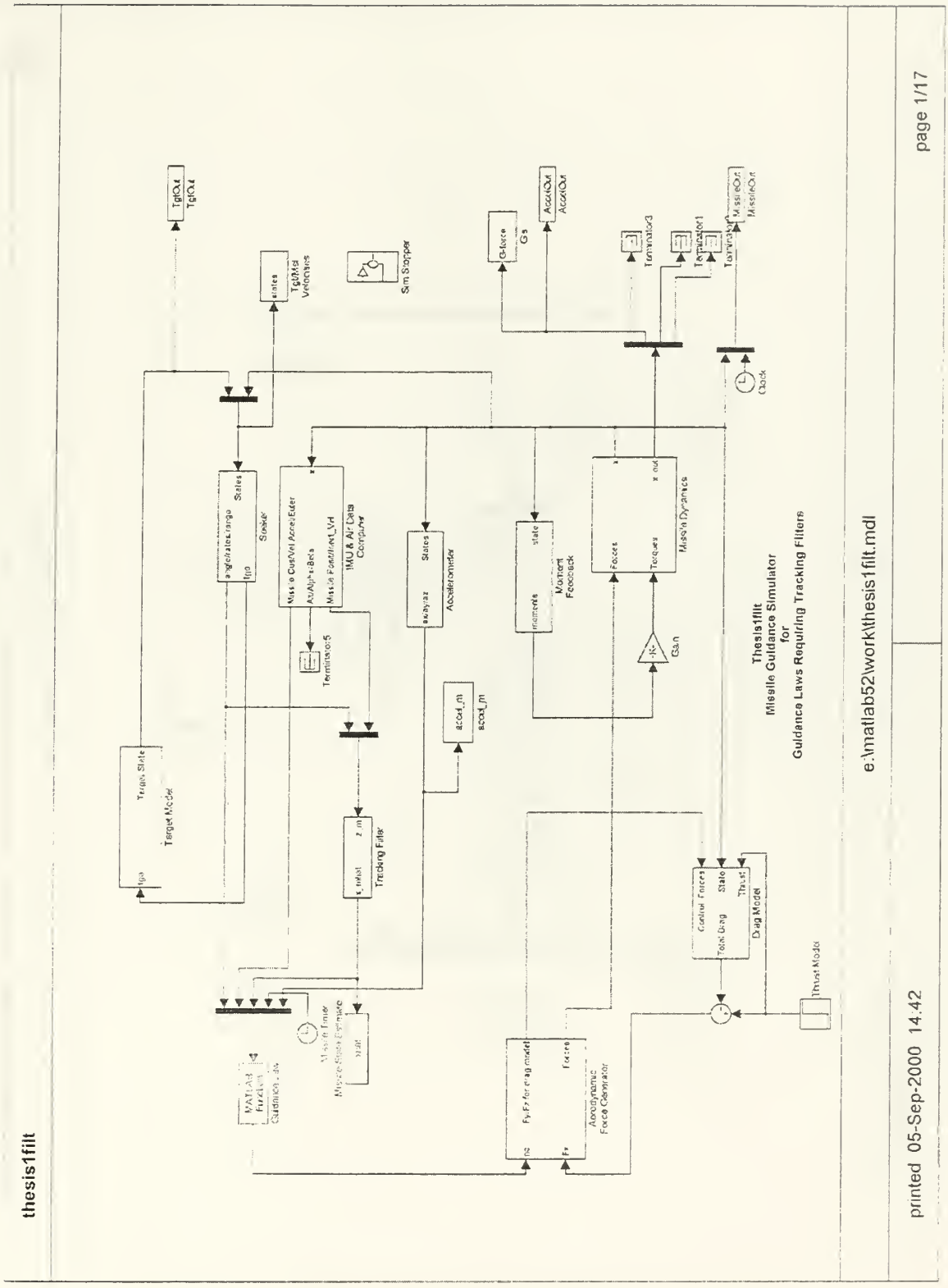


Figure A.14. Simplified 6DOF model with filter.

Figure A.15. Accelerometer.

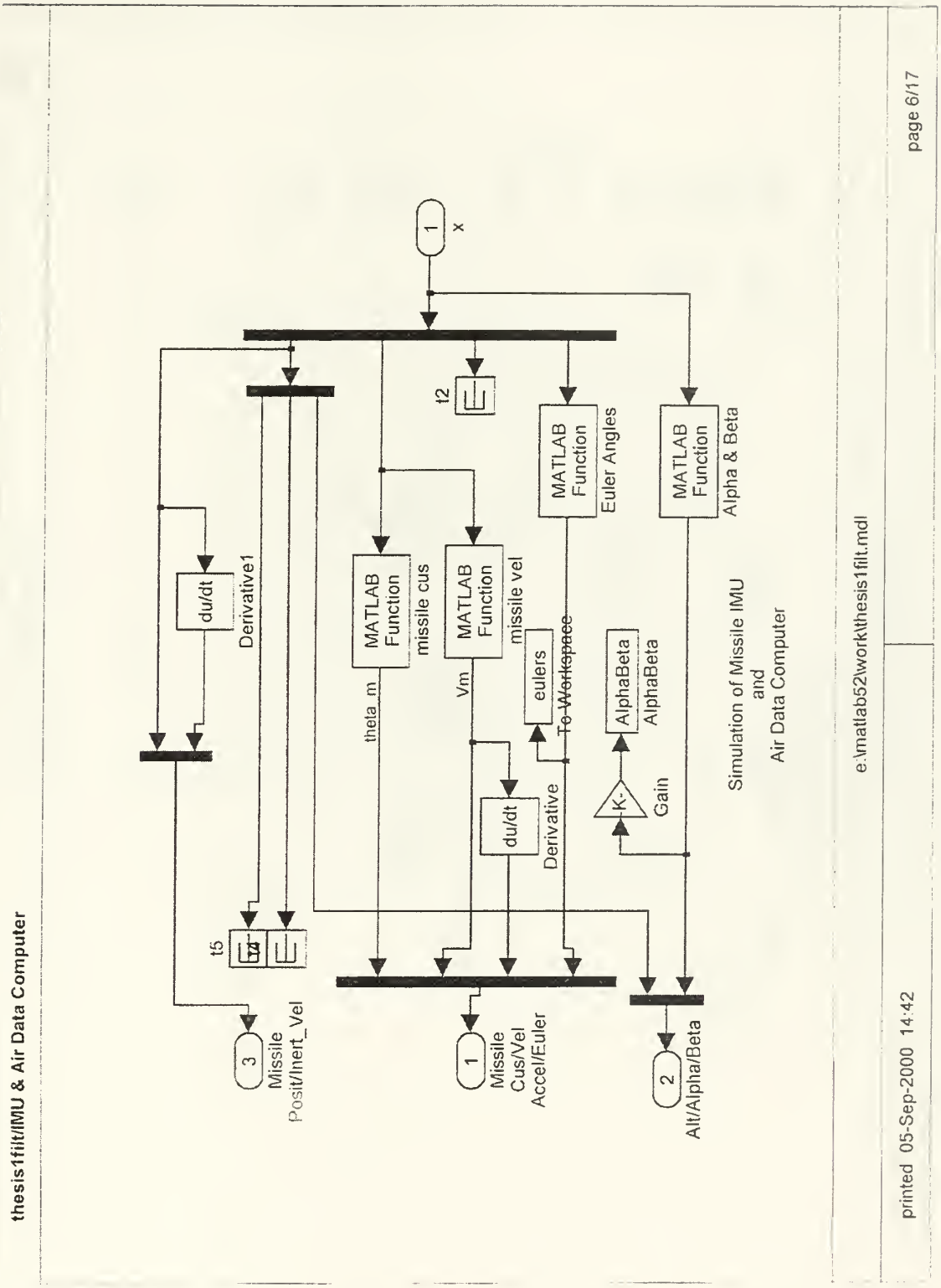


Figure A.16. IMU with additional outputs for tracking filter.

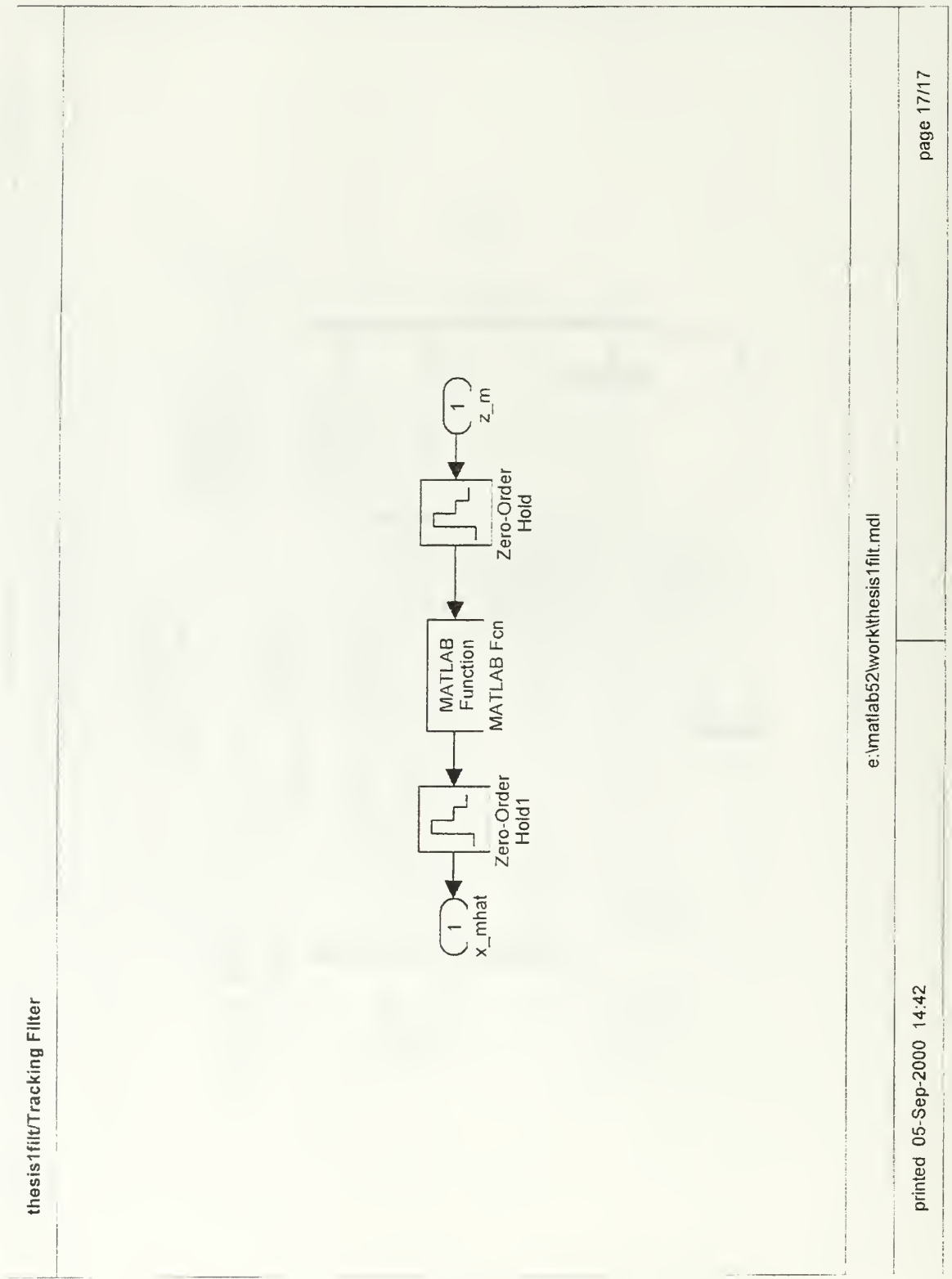


Figure A.17. α–β–γ tracking filter. Function block is ABGFILTER.M.

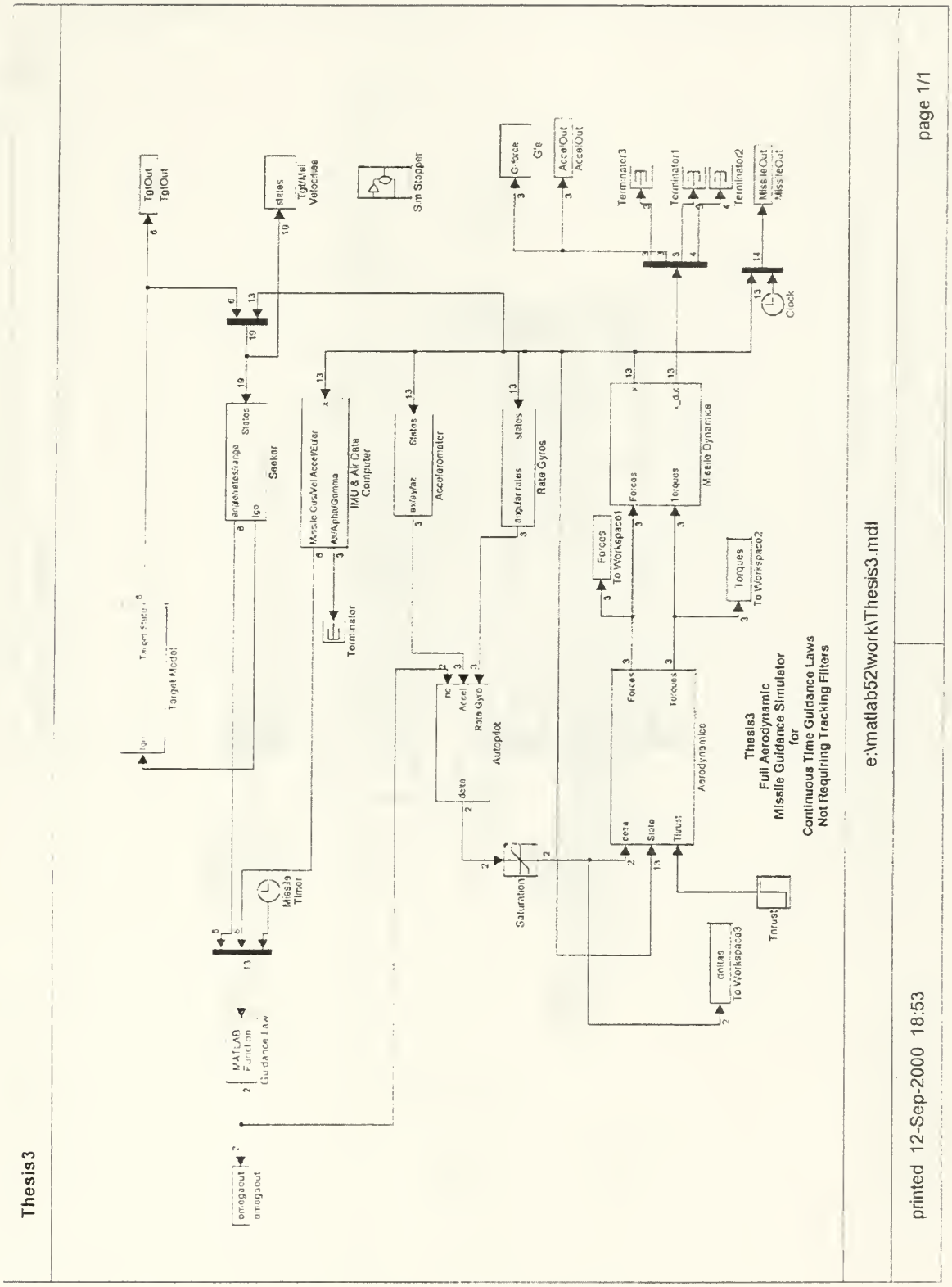


Figure A.18. Full aerodynamic 6DOF model.

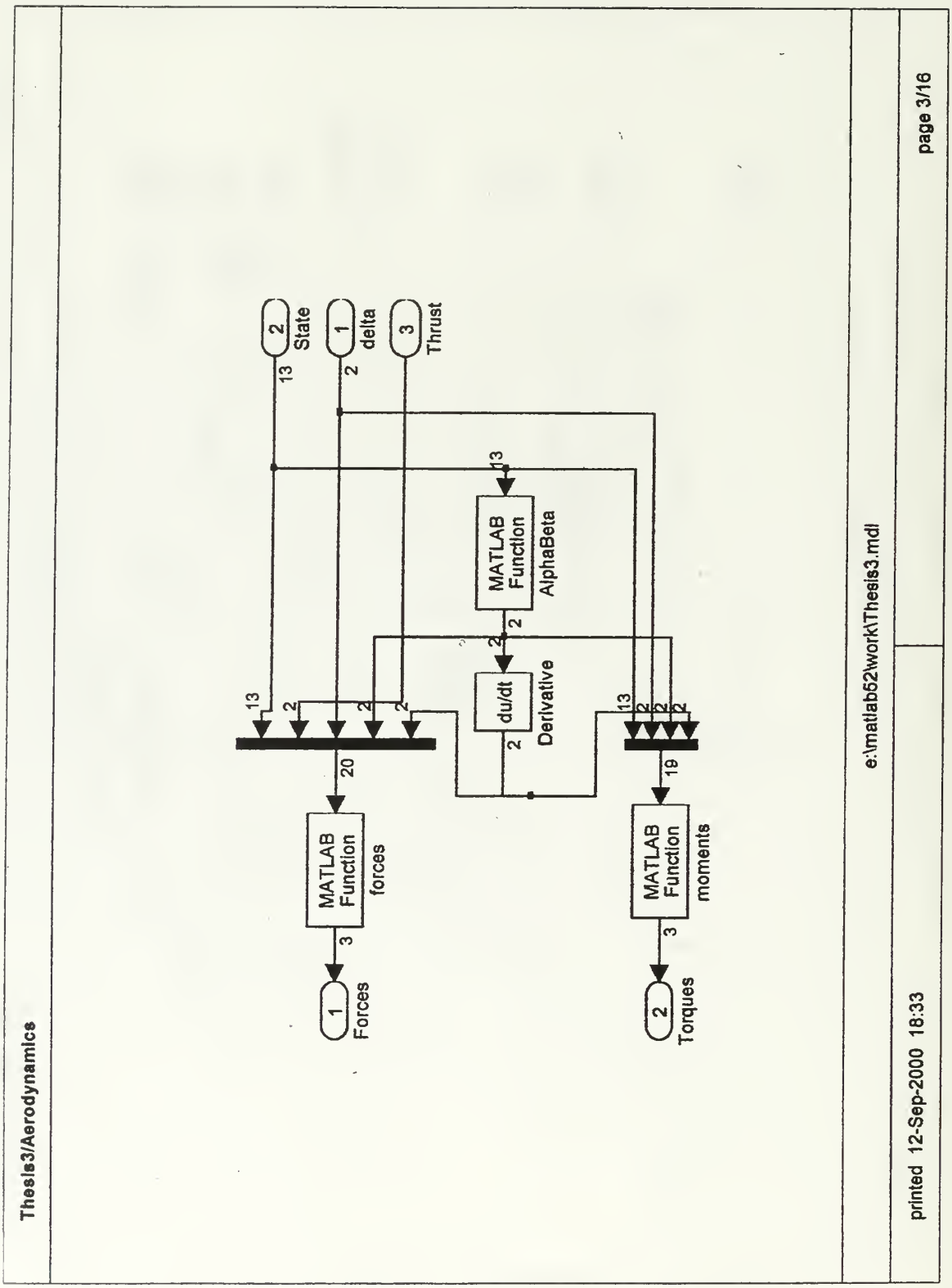


Figure A.19. Aerodynamic moment and force models. Function blocks are ALPHABETA.M, AEROFORCES.M, and AEROMOMENTS.M.

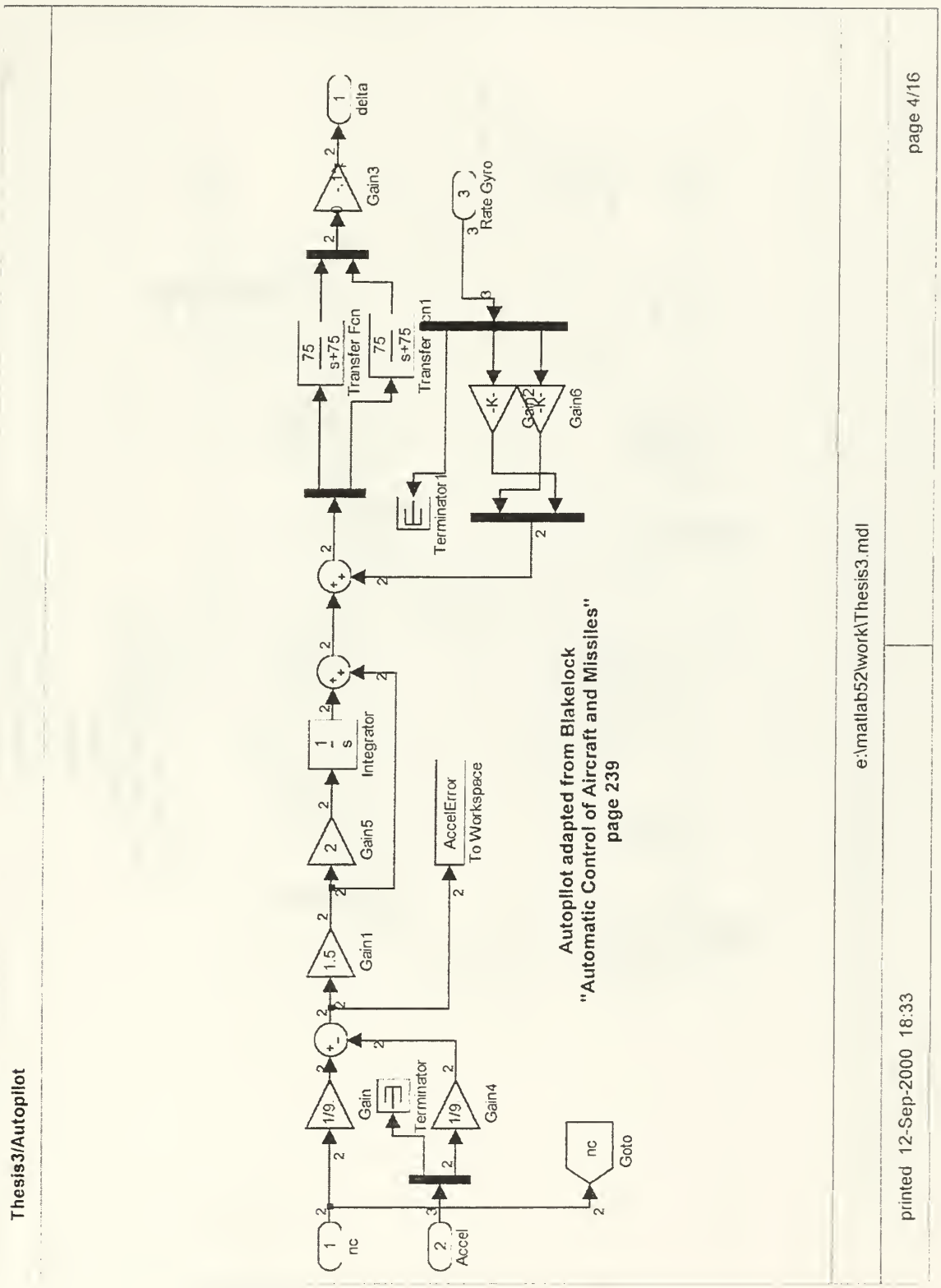


Figure A.20. Full aero model autopilot.

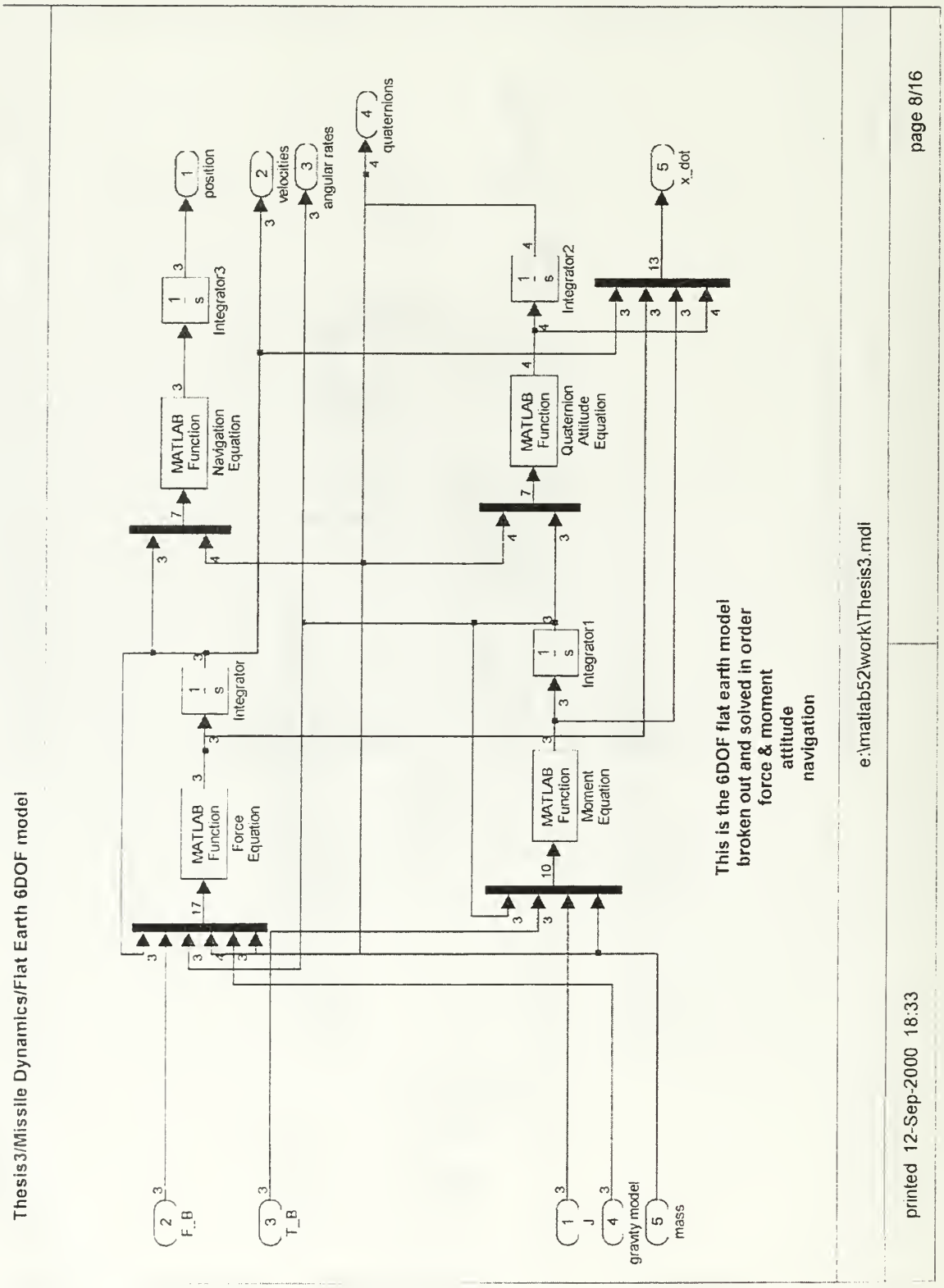


Figure A.21. Full aero model 6DOF equations. Function blocks are EQNFORCE.M, EQNMOMENT.M, EQNQUAT.M, and EQNPOS.M

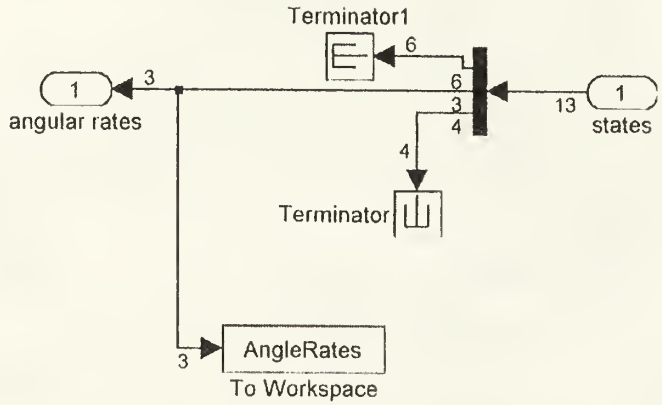


Figure A.22. Rate gyros.

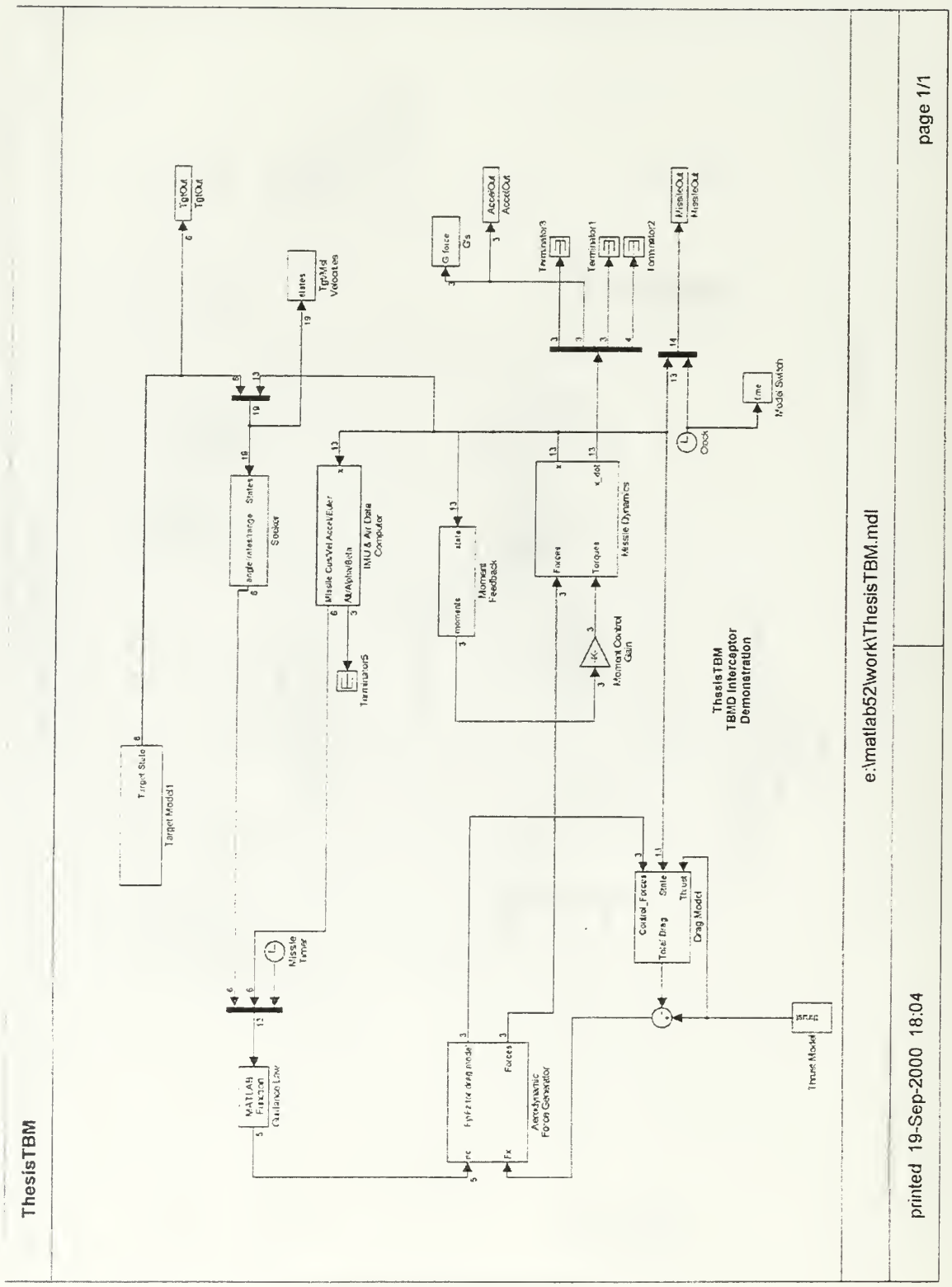


Figure A.23. Simplified 6DOF TBMD interceptor simulation.

Figure A.24. TBM thrust model.

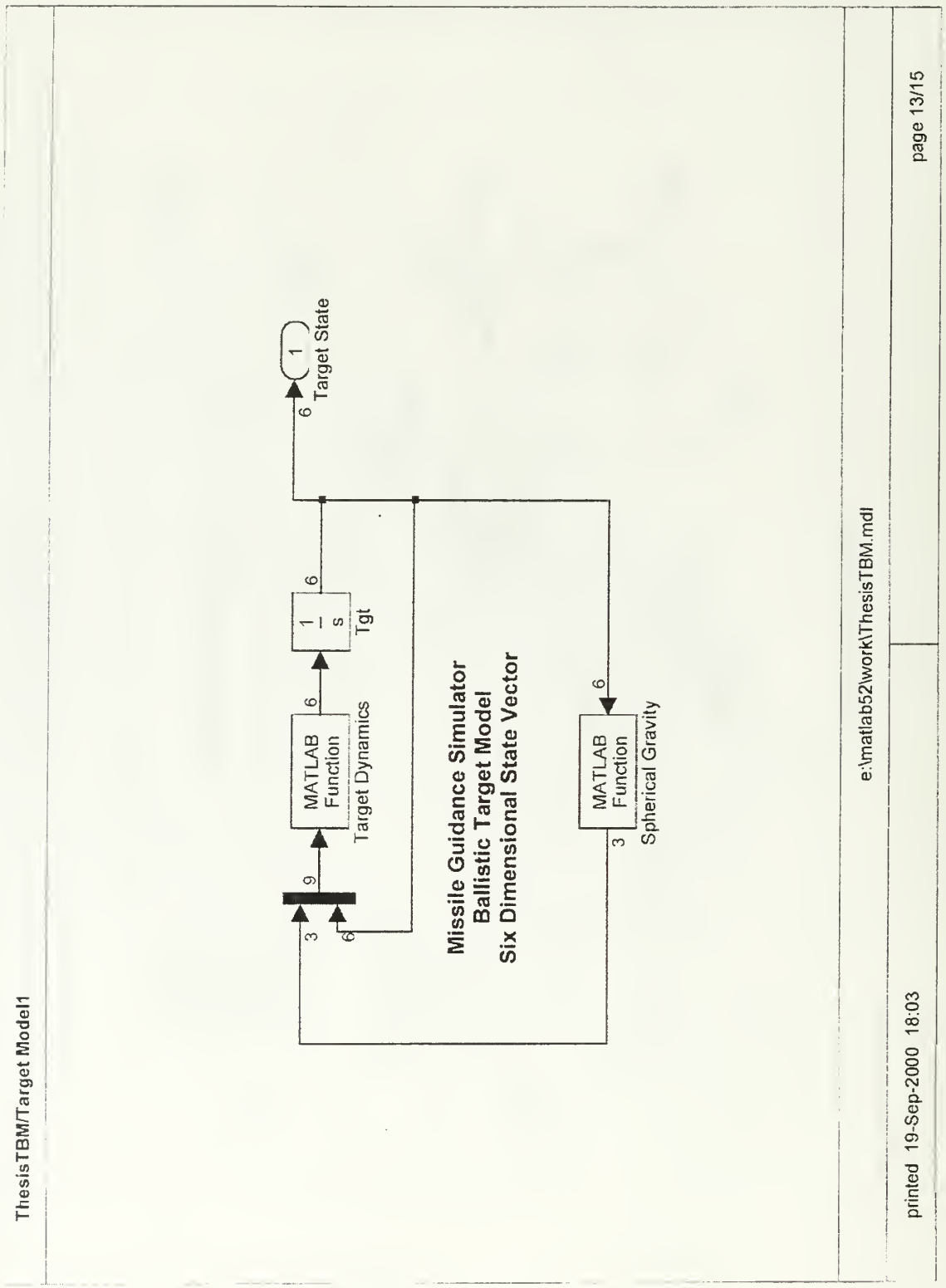


Figure A.25. TBM target model. Function blocks are BALLISTDYN.M and GRAVITY.M.

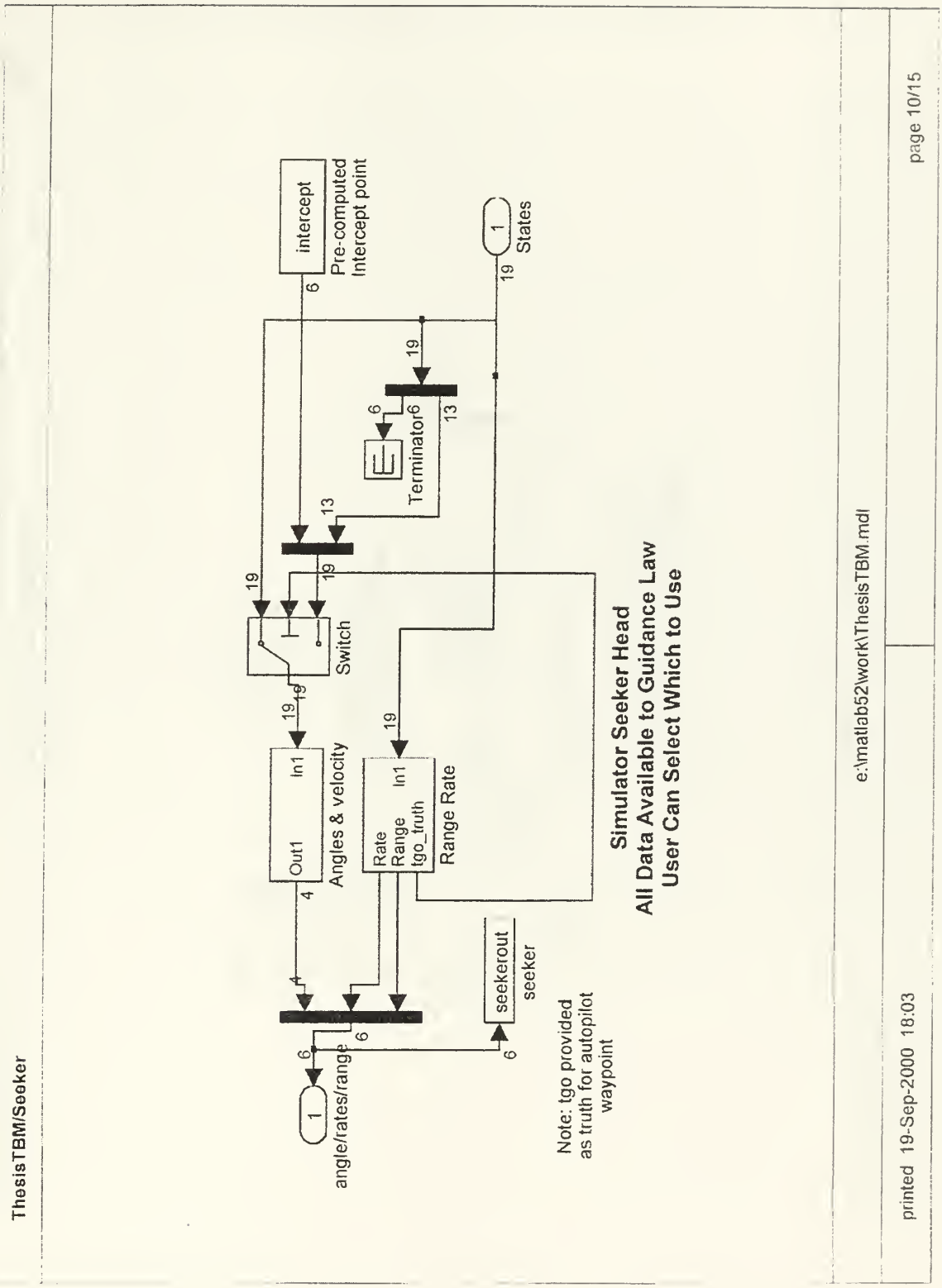


Figure A.26. TBM seeker model.

Figure A.27. TBM missile model switch. Function block is MODELSWITCH.M.

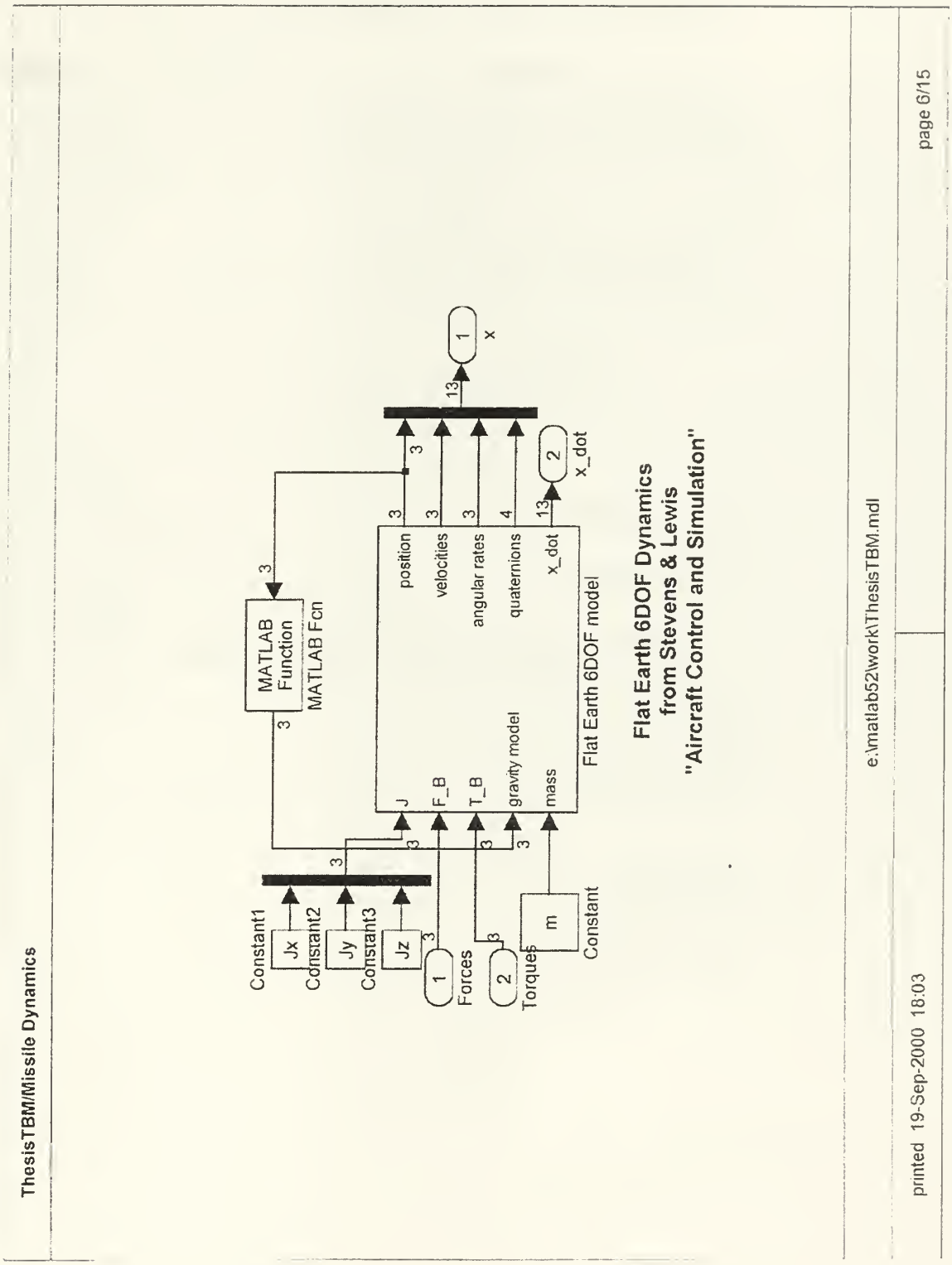


Figure A.28. TBM Missile Dynamics. Function block is GRAVITY2.M.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B. MATLAB® CODE

Filename	Purpose
abgfilter.m	implements alpha-beta-gamma filter
aeroforces.m	aerodynamic forces for full aero model
aeromoments.m	aerodynamic moments for full aero model
alphabeta.m	angles of attack
auxplots.m	plots auxiliary data for one simulation run
b2quat.m	quaternions from a rotation matrix
bangpt.m	bang-bang control law for 6DOF model
bryson.m	DG control law
cd0.m	parasitic drag coefficient
cdi.m	induced drag coefficient
cdvmach	polyfit for cdi
chingfanlin	APN guidance law
draginduced	induced drag force
draginducedtbm	induced drag force for TBM simulation
dragthesis	parasitic drag force
dragthesistbm	parasitic drag force for TBM simulation
drawmissile	missile plan view
dynamic3d	3D target dynamics
eqnforce	force dynamics for full aero model
eqnmoment	moment dynamics for full aero model
eqnposit	navigation equation for full aero model
eqnquat	quaternion dynamics for full aero model
flatearthdyn	6DOF dynamics for flat earth model
formdrag	computes form drag
gravity	spherical earth gravity for TBM target
gravity2	spherical earth gravity for TBM interceptor
kbfilter	kinematic boundary for filtered laws
kbouter2	kinematic boundary for unfiltered laws
machvalt	computes Mach 1 at altitude
missiledata	data for AMRAAM
missiledata2	data for JERGER
missiledata3	data for SM-2 MR
missiledata4	data for SM-2 ER
modelswitch	switches models at staging in TBM simulation
noisestudy	noise study with 100 realizations
propnav3d	PN law for full aero model
propnavpt	PN law for 6DOF model
propnavtbm	PN law for TBM simulation

Table B.1. Matlab® Source Code Listing.

Filename	Purpose
q2euler	computes euler angles from quaternions
quat2b	computes rotation matrix from quaternions
quaternion	computes quaternion from euler angles
rhovalt	computes atmospheric density
sixdofdyn	6DOF dynamics in ECI coordinates
spielberg	movie maker
tgo	computes time to go
tgtset	initializes target for AAM simulations
thebigstop	simulation stopper
thesis2plot	plots data for thesis
thesisinit	initializes simulator
vcpropnavpt	VCPN law

Table B.1. Matlab® Source Code Listing (continued)

```
function y=abgfilter(u)
%ABGFILTER Implements an alpha-beta-gamma filter as
%          outlined in Bar-Shalom & Li "Estimation and
%          Tracking"
%          see also
%          Copyright 1999-2000 by Triple B Enterprises

%//*****************************************************************
*********
%// File:  abgfilter.m
%// Name:  LCDR Robert D. Broadston
%// MSEE/EE Thesis
%// Operating Environment:  Windows NT 4.0 Service Pack 5
%// Compiler:  MatLab v5.3
%// Date:  19 July 2000
%// Description:  Implements a 9-dimensional state vector
%//               alpha-beta-gamma tracking filter for use with
%//               missile guidance laws requiring tracking filters
%//               Note:  uses global XLAST to preserve state
%//               between iterations
%// Inputs:  measurements (los,los_dot,R,R_dot),
%//          missile pos (x,y,z)
%// Outputs:  9-dimensional estimate of target state
%//           [x,vx,ax,y,vy,ay,z,vz,az]'
%// Process:  alpha-beta-gamma filter outlined in Bar-Shalom & Li
%// Assumptions:
%// Warnings:  may require up to 20 samples to stabilize from
%//            initialization
%//*****************************************************************
*********
%// Order of elements
%//      -Define globals
%//      -Define constants
%//      -Define elements of input vector
%//      -Functions
%//*****************************************************************
*********
% ******  define globals  ******
global m d L XCG XCPN XCPW XCPB XHL
global ST SW SPLAN SREF FILTSAMP XLAST

% ******  define constants  ******

% ******  define input vector  ******
losdot=u(1);
phidot=u(2);
los=u(3);
phi=u(4);
rdot=u(5);
R=u(6);
xm=u(7);
ym=u(8);
zm=u(9);
```

```
% ******  initialize variables  ******
% compute target cartesian coordinates
xt=R*cos(los)+xm;
yt=R*sin(los)+ym;
zt=R*sin(phi)+zm;

z=[xt;yt;zt];

% set noise parameters
sigmav=1;
sigmaw=1;
lamda=sigmav*FILTSAMP^2/sigmaw;

% set filter parameters from Bar-Shalom & Li
falpha=.9;
fbeta=.9;
fgamma=.9;

% filter matrices
F=[1 FILTSAMP FILTSAMP^2/2 zeros(1,6);
   0   1      FILTSAMP   zeros(1,6);
   0   0          1      zeros(1,6);
   zeros(1,3)           1 FILTSAMP FILTSAMP^2/2 zeros(1,3);
   zeros(1,4)              1      FILTSAMP   zeros(1,3);
   zeros(1,5)                  1             zeros(1,3);
   zeros(1,6)                     1 FILTSAMP FILTSAMP^2/2;
   zeros(1,7)                         1      FILTSAMP;
   zeros(1,8)                                    1];
H=[1 0 0 0 0 0 0 0 0;
   0 0 0 1 0 0 0 0 0;
   0 0 0 0 0 0 1 0 0];

% compute steady state gains
W=[falpha;fbeta/FILTSAMP;fgamma/(2*FILTSAMP^2)];

% build gain matrix
P=[W            zeros(3,2);
   zeros(3,1) W zeros(3,1);
   zeros(3,2)       W];

% ************  functions  ************

% run filter
xhat=F*XLAST;
xhat1=xhat+P*(z-H*xhat);

XLAST=xhat1;

y=xhat1;

%//end of file abgfilter.m
```

```
function y=aeroforces(u)
%AEROFORCES Computes aerodynamic forces on a missile.
%           derived from Zarchan "Tactical and Strategic
%           Missile Guidance" and Anderson "Fundamentals
%           of Aerodynamics"
%           see also AEROMOMENTS
%           Copyright 1999-2000 by Triple B Enterprises

%//*****************************************************************
*********
%// File:  aeroforces.m
%// Name:  LCDR Robert D. Broadston
%// MSEE/EE Thesis
%// Operating Environment:  Windows NT 4.0 Service Pack 5
%// Compiler:  MatLab v5.3
%// Date:  7 Sept 2000
%// Description:  Computes aerodynamic forces for both subsonic
%//               and supersonic regimes on a symmetrical STT
%//               missile.
%// Inputs:  missile state, control deflections, angles of attack
%//          and rates
%// Outputs:  Body centered aerodynamic force components [Fx,Fy,Fz]'
%// Process:  Brute force computation of equations from Zarchan and
%//           Anderson
%// Assumptions:
%// Warnings:
%//*****************************************************************
*********
%// Order of elements
%//      -Define globals
%//      -Define constants
%//      -Define elements of input vector
%//      -Functions
%//*****************************************************************
*********
% ******  define globals  ******
global m d L XCG XCPN XCPW XCPB XHL
global ST SW SPLAN SREF

% ******  define constants  ******

% ******  define input vector  ******
states=u(1:13);
delta_r=u(14);
delta_e=u(15);
thrust=u(16);
m_alpha=u(17);
m_beta=u(18);
alphadot=u(19);
betadot=u(20);
```

```
% ******  initialize variables  ******
V_m=sqrt(u(4)^2+u(5)^2+u(6)^2);                % missile velocity
Mach=V_m/machvalt(u(3));                       % Mach number
M_BETA=sqrt(Mach^2-1);                         % Beta factor
Q=rhovalt(u(3))*V_m^2/2;                       % dynamic pressure

% ************  functions  ************
%----------- compute normal coefficients -------------------
%-------- these equations developed in Zarchan -------------
if (Mach>1.05)
   C_Naz=2+3*SPLAN*m_alpha/(2*SREF)...
      +8*SW/(M_BETA*SREF)...
      +8*ST/(M_BETA*SREF);
   C_Ndz=8*ST/(M_BETA*SREF);
   C_Nz=C_Naz*m_alpha+C_Ndz*delta_e;
   C_Nby=2+3*SPLAN*m_beta/(2*SREF)...
      +8*SW/(M_BETA*SREF)...
      +8*ST/(M_BETA*SREF);
   C_Ndy=8*ST/(M_BETA*SREF);
   C_Ny=C_Nby*m_beta+C_Ndy*delta_r;
%------- these equations developed in Anderson -------------
else
   C_Nz=.5*m_alpha;
   C_Ny=.5*m_beta;
end

%------------ compute drag --------------------
CD0=cd0([states;thrust]);                      % drag
CDI=cdi([C_Nz,C_Ny,m_alpha,m_beta,u(3),V_m]);  % coefficients

drag=(CDI+CD0)*Q*SREF;

%------------ compute forces ------------------
F_x=0;%thrust-drag;
F_y=C_Ny*Q*SREF;
F_z=C_Nz*Q*SREF;

y=[F_x;F_y;F_z];

%//end of file aeroforces.m
```

```
function y=aeromoments(u)
%AEROMOMENTS Computes aerodynamic moments on a missile.
%            derived from Zarchan "Tactical and Strategic
%            Missile Guidance" and Anderson "Fundamentals
%            of Aerodynamics"
%            see also AEROFORCES
%            Copyright 1999-2000 by Triple B Enterprises

%//*********************************************************************
*********
%// File:  aeromoments.m
%// Name:  LCDR Robert D. Broadston
%// MSEE/EE Thesis
%// Operating Environment:  Windows NT 4.0 Service Pack 5
%// Compiler:  MatLab v5.3
%// Date:  7 Sept 2000
%// Description:  Computes aerodynamic moments for both subsonic
%//               and supersonic regimes on a symmetrical STT
%//               missile.  Note:  Moment about x-axis is
%//               negative feedback of roll rate to stop missile
%//               from rolling.
%// Inputs:  missile state, control deflections, angles of attack
%//          and rates
%// Outputs:  Body centered aerodynamic moments [Tx,Ty,Tz]'
%// Process:  Brute force computation of equations from Zarchan and
%//           Anderson
%// Assumptions:
%// Warnings:
%//*********************************************************************
*********
%// Order of elements
%//      -Define globals
%//      -Define constants
%//      -Define elements of input vector
%//      -Functions
%//*********************************************************************
*********
% ******  define globals  ******
global m d L XCG XCPN XCPW XCPB XHL
global ST SW SPLAN SREF

% ******  define constants  ******

% ******  define input vector  ******
states=u(1:13);
delta_r=u(14);
delta_e=u(15);
m_alpha=u(16);
m_beta=u(17);
alphadot=u(18);
betadot=u(19);
```

```
% ******  initialize variables  ******
V_m=sqrt(u(4)^2+u(5)^2+u(6)^2);                % missile velocity
Mach=V_m/machvalt(u(3));                       % Mach number
M_BETA=sqrt(Mach^2-1);                         % Beta factor
Q=rhovalt(u(3))*V_m^2/2;                       % dynamic pressure


% ************  functions  ************
%----------- compute moment coefficients --------------------
%------these equations developed in Zarchan
if Mach>1.05
   C_My=2*(XCG-XCPN)/d...
      +3*SPLAN*m_alpha*(XCG-XCPB)/(2*SREF*d)...
      +8*SW*(XCG-XCPW)/(M_BETA*SREF*d)...
      +8*ST*(XCG-XHL)/(M_BETA*SREF*d);
   C_Mdy=8*ST*(XCG-XHL)/(M_BETA*SREF*d);
   C_Mz=2*(XCG-XCPN)/d...
      +3*SPLAN*m_beta*(XCG-XCPB)/(2*SREF*d)...
      +8*SW*(XCG-XCPW)/(M_BETA*SREF*d)...
      +8*ST*(XCG-XHL)/(M_BETA*SREF*d);
   C_Mdz=8*ST*(XCG-XHL)/(M_BETA*SREF*d);
%------ these equations developed in Anderson
else
   C_My=.5;  C_Mdy=.05;
   C_Mz=.5;  C_Mdz=.05;
end

T_x=-400*states(7);
T_y=(C_My*m_alpha+C_Mdy*delta_e)*Q*SREF*d-800*alphadot;
T_z=(-C_Mz*m_beta+C_Mdz*delta_r)*Q*SREF*d+800*betadot;

y=[T_x;T_y;T_z];

%//end of file aeromoments.m
```

```
function y=alphabeta(u)
%ALPHABETA Computes angles of attack in both vertical
%          and horizontal planes
%          see also
%          Copyright 1999-2000 by Triple B Enterprises

%//*****************************************************************
*********
%// File:  projX.m
%// Name:  LCDR Robert D. Broadston
%// MSEE/EE Thesis
%// Operating Environment:  Windows NT 4.0 Service Pack 5
%// Compiler:  MatLab v5.3
%// Date:  31 July 2000
%// Description:  Computes angles of attack using ATAN formulation
%//               in Bryson "Control of Spacecraft and Aircraft"
%// Inputs:  missile state
%// Outputs:  angles of attack [alpha,beta]'
%// Process:  ATAN formulation of Bryson
%// Assumptions:
%// Warnings:
%//*****************************************************************
*********
%// Order of elements
%//         -Define globals
%//         -Define constants
%//         -Define elements of input vector
%//         -Functions
%//*****************************************************************
*********
% ******  define globals  ******

% ******  define constants  ******

% ******  define input vector  ******
v=[u(4);u(5);u(6)];
% ******  initialize variables  ******

% ************  functions  ************
%---------- these equations developed in Bryson -------
% using beta1 for sideslip angle to avoid problems with
% built-in matlab fxn beta

alpha=atan2(v(3),sqrt(v(1)^2+v(2)^2));
beta1=atan2(v(2),v(1));

y=[alpha;beta1];

%//end of file alphabeta.m
```

```
%//*************************************************************************
*********
%// File:  auxplots.m
%// Name:  LCDR Robert D. Broadston
%// MSEE/EE Thesis
%// Operating Environment:  Windows NT 4.0 Service Pack 5
%// Compiler:  MatLab v5.3
%// Date:  13 April 2000
%// Description:  Plots auxiliary variables from missile simulations
%// Inputs:  none
%// Outputs:  plots of auxiliary variables
%// Process:  none
%// Assumptions:  none
%// Warnings:  none
%//*************************************************************************
*********
%// Order of elements
%//      -Define globals
%//      -Define constants
%//      -Define elements of input vector
%//      -Functions
%//*************************************************************************
*********
% ******  define globals  ******

% ******  define constants  ******

% ******  define input vector  ******

% ******  initialize variables  ******

% ************  functions  ************
figure(4)
subplot(4,2,1)
plot(t,AccelOut/9.8045)
ylabel('AccelOut')
subplot(4,2,2)
plot(t,AlphaBeta)
ylabel('AlphaBeta')
subplot(4,2,3)
plot(t,eulers*57.3)
ylabel('eulers')
subplot(4,2,4)
plot(t,MissileV)
ylabel('MissileV')
subplot(4,2,5)
plot(t,AccelError)
ylabel('AccelError')
subplot(4,2,6)
plot(t,seeker)
ylabel('seeker')
subplot(4,2,7)
plot(t,deltas)
ylabel('deltas')
%//end of file auxplots.m
```

```
function y=b2quat(B)
%B2QUAT  Computes quaternions from a rotation matrix
%        B2QUAT(B)
%        see also QUATERNION, BQUAT
%        Copyright 1999-2000 by Triple B Enterprises

%//*********************************************************************
*********
%// File:  b2quat.m
%// Name:  LCDR Robert D. Broadston
%// MSEE/EE Thesis
%// Operating Environment:  Windows NT 4.0 Service Pack 5
%// Compiler:  MatLab v5.3
%// Date:  12 Dec 1999
%// Description:  Computes quaternions from ABC rotation matrix
%//               using formulation of Kuiper "Quaternions and
%//               Rotation Sequences"
%// Inputs:  rotation matrix B
%// Outputs:  quaternion [q0,q1,q2,q3]'
%// Process:  Kuiper pp. 166-167
%// Assumptions:
%// Warnings:
%//*********************************************************************
*********
%// Order of elements
%//      -Define globals
%//      -Define constants
%//      -Define elements of input vector
%//      -Functions
%//*********************************************************************
*********
% ******  define globals  ******

% ******  define constants  ******

% ******  define input vector  ******

% ******  initialize variables  ******

% ************  functions  ************
q_0=sqrt((1+B(1,1)+B(2,2)+B(3,3))/4);
q_1=sqrt((1+B(1,1)-B(2,2)-B(3,3))/4);
q_2=sqrt((1-B(1,1)+B(2,2)-B(3,3))/4);
q_3=sqrt((1-B(1,1)-B(2,2)+B(3,3))/4);

a=(B(2,3)-B(3,2))/4;
b=(B(3,1)-B(1,3))/4;
c=(B(1,2)-B(2,1))/4;
d=(B(1,2)+B(2,1))/4;
e=(B(2,3)+B(3,2))/4;
f=(B(1,3)+B(3,1))/4;
```

```
if (a<0 & b<0 & c<0)
   q_0=-q_0;
end
if (a<0 & d<0 & f<0)
   q_1=-q_1;
end
if (b<0 & d<0 & e<0)
   q_2=-q_2;
end
if (c<0 & e<0 & f<0)
   q_3=-q_3;
end

y=[q_0;q_1;q_2;q_3];

%//end of file b2quat.m
```

```
function y=bangpt(u)
%BANGPT Computes bang-bang control law for  simplified
%       6DOF model
%       see also PROPNAVPT, VCPROPNAVPT, BRYSON, CHINGFANLIN
%       Copyright 1999-2000 by Triple B Enterprises

%//**********************************************************************
*********
%// File:  BANGPT.m
%// Name:  LCDR Robert D. Broadston
%// MSEE/EE Thesis
%// Operating Environment:  Windows NT 4.0 Service Pack 5
%// Compiler:  MatLab v5.3
%// Date:  6 Aug 2000
%// Description:  Bang-bang control law for 6DOF flight model.
%//               Uses two values of bang depending on range to
%//               reduce problems with drag at start of engagement.
%//               -.005 rad/s dead band on los rate
%// Inputs:  [seeker data,IMU data,timer]
%// Outputs:  [command accelerations,applied forces]
%// Process:  bang-bang control law
%// Assumptions:  none
%// Warnings:  none
%//**********************************************************************
*********
%// Order of elements
%//      -Define globals
%//      -Define constants
%//      -Define elements of input vector
%//      -Functions
%//**********************************************************************
*********
% ******  define globals  ******
global m satflag

% ******  define constants  ******
Nprime=5;
Nprimez=5;

% ******  define input vector  ******
thetadot=u(1);
phidot=u(2);
los=u(3);
philos=u(4);
R=u(6);
Vc=-u(5);
heading=u(7);
Vm=u(8);
Vmdot=u(9);
phi=u(10);
theta=u(11);
psi=u(12);
time=u(13);
```

```
% ******  initialize variables  ******
% set max control force to 5 g (long range
% & 20 g (end game)
if (R>5000)
   Nbang=5*9.8045;
else
   Nbang=20*9.8045;
end

% ************  functions  ************
% establish a dead band on theta dot
if (abs(thetadot)>.01*pi/180)
   ny=Nbang*sign(Vc*thetadot);
else
   ny=0;
end

nz=Nprimez*Vc*(phidot)-9.8045;

% control force limiter
if satflag
   if (abs(ny)>30*9.8045)
      ny=sign(ny)*30*9.8045;
   end
   if (abs(nz)>30*9.8045)
      nz=sign(nz)*30*9.8045;
   end
end

% compute ABC forces applied
Fx=0;
Fy=ny*m;
Fz=nz*m;

% output vector
y=[ny;nz;Fx;Fy;Fz];

%//end of file BANGPT.m
```

```
function y=bryson(u)
%BRYSON Computes optimal guidance law derived by Bryson & Ho
%       with dragforce inputs for point mass simulation
%       see also PROPNAVPT BANGPT CHINGFANLIN
%       Copyright 1999-2000 by Triple B Enterprises
%//*********************************************************************
*********
%// File:  bryson.m
%// Name:  LCDR Robert D. Broadston
%// MSEE/EE Thesis
%// Operating Environment:  Windows NT 4.0 Service Pack 5
%// Compiler:  MatLab v5.3
%// Date:  18 Sep 00
%// Description:  Modified PN differential games guidance
%//               law from Bryson & Ho
%// Inputs:  Seeker outputs, filter outputs, missile timer
%//          accelerometer output
%// Outputs:  command accelerations, y and z forces for drag model
%// Process:
%// Assumptions:
%// Warnings:
%//*********************************************************************
*********
%// Order of elements
%//      -Define globals
%//      -Define constants
%//      -Define elements of input vector
%//      -Functions
%//*********************************************************************
*********
% ******  define globals  ******
global m satflag

% ******  define constants  ******
Nprime=3;
Nprimez=3;

% ******  define input vector  ******
thetadot=u(1);
phidot=u(2);
los=u(3);
philos=u(4);
R=u(6);
Vc=-u(5);
heading=u(7);
Vm=u(8);
Vmdot=u(9);
phi=u(10);
theta=u(11);
psi=u(12);

m_state=u(13:21);
time=u(22);

accel_in=u(23:25);
```

```
% ******  initialize variables  ******

% ************  functions  ************
if time<2.0
   ny=Nprime*Vc*(thetadot)/cos(psi-los);
   nz=Nprimez*Vc*(phidot)/cos(theta-philos)-9.8045;
else
   cp=30*9.8045;
   ce_lat=sqrt(m_state(3)^2+m_state(6)^2);

   if (ce_lat==cp)
      ce_lat=29*9.8045;
   end

   ce_vert=abs(m_state(9));

   if (ce_vert==cp)
      ce_vert=29*9.8045;
   end

   ny=3/(1-ce_lat/cp)*Vc*thetadot;
   nz=3/(1-ce_vert/cp)*Vc*phidot-9.8045;
end

if satflag
   if (abs(ny)>30*9.8045)
      ny=sign(ny)*30*9.8045;
   end
   if (abs(nz)>30*9.8045)
      nz=sign(nz)*30*9.8045;
   end
end



Fx=0;
Fy=ny*m;
Fz=nz*m;

y=[ny;nz;Fx;Fy;Fz];

%//end of file bryson.m
```

```
function y=cd0(u);
% CDO Computes induced drag coefficient
%
%        see also CDI
%        Copyright 1999-2000 by Triple B Enterprises
%//*****************************************************************
*********
%// File:  cd0.m
%// Name:  LCDR Robert D. Broadston
%// MSEE/EE Thesis
%// Operating Environment:  Windows NT 4.0 Service Pack 5
%// Compiler:  MatLab v5.3
%// Date:  9 May 00
%// Description:  computes induced drag coefficient for full
%//               aero model
%// Inputs:  state, boost status
%// Outputs:  drag coeffient
%// Process:  polynomial fit to data from Hutchins EC4330 notes
%// Assumptions:
%// Warnings:
%//*****************************************************************
*********
%// Order of elements
%//      -Define globals
%//      -Define constants
%//      -Define elements of input vector
%//      -Functions
%//*****************************************************************
*********
% ******  define globals  ******

% ******  define constants  ******
NoBoost=[-0.0014    0.0299   -0.2110    0.6256];
Boost=[-0.0012    0.0243   -0.1521    0.4044];

% ******  define input vector  ******
v=sqrt(u(4)^2+u(5)^2+u(6)^2);
alt=u(3);

boost=u(14);

% ******  initialize variables  ******
mach=v/machvalt(alt);

% ************  functions  ************
if (mach>100)
   mach=.83;
end
```

```
% these curves approximated from "typical" data presented
% in Stevens & Lewis and Hutchins

% compute Cd0
if (boost & (mach<1))
   y=.15;
end

if (~boost & (mach<1))
   y=.25;
end

if ((mach>=1) & (boost~=0))
   y=polyval(Boost,mach);
end

if ((mach>=1) & (boost==0))
   y=polyval(NoBoost,mach);
end

if ((mach>5) & boost)
   y=.10;
end

if ((mach>6.4) & ~boost)
   y=.132;
end

%//end of file cd0.m
```

```
function y=cdi(u)
%CDI    Computes induced drag coefficient
%        see also CD0
%        Copyright 1999-2000 by Triple B Enterprises
%//*********************************************************************
*********
%// File:  cdi.m
%// Name:  LCDR Robert D. Broadston
%// MSEE/EE Thesis
%// Operating Environment:  Windows NT 4.0 Service Pack 5
%// Compiler:  MatLab v5.3
%// Date:  18 Apr 00
%// Description:  computed induced drag coefficient for full
%//               aero model
%// Inputs:  see below
%// Outputs:  cdi
%// Process:  Anderson
%// Assumptions:
%// Warnings:
%//*********************************************************************
*********
%// Order of elements
%//      -Define globals
%//      -Define constants
%//      -Define elements of input vector
%//      -Functions
%//*********************************************************************
*********
% ******  define globals  ******

% ******  define constants  ******

% ******  define input vector  ******
C_Nz=u(1);
C_Ny=u(2);
m_alpha=u(3);
m_beta=u(4);
alt=u(5);
v=u(6);
% ******  initialize variables  ******
Mach=v/machvalt(alt);

% ************  functions  ************
%-------- these equations developed from Anderson -------
if (Mach>1.0)
   M_BETA=sqrt(Mach^2-1);
   Cdi=(4*m_alpha^2/M_BETA+4*m_beta^2/M_BETA);
else
   Cdi=(C_Ny^2+C_Nz^2)/pi;
end

y=Cdi;

%//end of file cdi.m
```

```
function y=cdvmach(mach,boost)
%CDVMACH   Computes approximation of zero lift drag
%          coefficient vs. mach number
%        CDVMACH(MACH,BOOST)
%        see also MACHVALT
%        Copyright 1999-2000 by Triple B Enterprises
%//*****************************************************************
*********
%// File:  cdvmach.m
%// Name:  LCDR Robert D. Broadston
%// MSEE/EE Thesis
%// Operating Environment:  Windows NT 4.0 Service Pack 5
%// Compiler:  MatLab v5.3
%// Date:  17 Apr 00
%// Description:  computes polynomial fit for cd0 vs Mach number
%// Inputs:  mach # and boost status
%// Outputs:  cd0
%// Process:  Fit on data from Hutchins EC4330 notes
%// Assumptions:
%// Warnings:
%//*****************************************************************
*********
%// Order of elements
%//      -Define globals
%//      -Define constants
%//      -Define elements of input vector
%//      -Functions
%//*****************************************************************
*********
% ******  define globals  ******

% ******  define constants  ******
NoBoost=[-0.0014    0.0299   -0.2110    0.6256];
Boost=[-0.0012    0.0243   -0.1521    0.4044];

% ******  define input vector  ******

% ******  initialize variables  ******

% ************  functions  ************
if (boost & (mach<1))
   y=.15;
end

if (~boost & (mach<1))
   y=.25;
end

if ((mach>=1) & (boost~=0))
   y=polyval(Boost,mach);
end

if ((mach>=1) & (boost==0))
   y=polyval(NoBoost,mach);
end
```

```
if ((mach>5) & boost)
   y=.10;
end

if ((mach>6.4) & ~boost)
   y=.132;
end

%//end of file cdvmach.m
```

```
function y=chingfanlin(u)
%CHINGFANLIN Computes optimal guidance law derived by Ching Fan Lin pg.
475
%            with dragforce inputs for point mass simulation
%        see also EXACTPROPNAV2
%        Copyright 1999-2000 by Triple B Enterprises
%//*****************************************************************
*********
%// File:  chingfanlin.m
%// Name:  LCDR Robert D. Broadston
%// MSEE/EE Thesis
%// Operating Environment:  Windows NT 4.0 Service Pack 5
%// Compiler:  MatLab v5.3
%// Date:  15 Sep 00
%// Description:  computes APN guidance law from Ching Fan Lin
%// Inputs:  seeker output, filter output, accelerometer,
%//          missile timer
%// Outputs:  command accelerations, y and z forces for drag
%// Process:
%// Assumptions:
%// Warnings:
%//*****************************************************************
*********
%// Order of elements
%//      -Define globals
%//      -Define constants
%//      -Define elements of input vector
%//      -Functions
%//*****************************************************************
*********
% ******  define globals  ******
global m satflag
% ******  define constants  ******
Nprime=3;
Nprimez=3;

% ******  define input vector  ******
thetadot=u(1);
phidot=u(2);
los=u(3);
philos=u(4);
R=u(6);
Vc=-u(5);
heading=u(7);
Vm=u(8);
Vmdot=u(9);
phi=u(10);
theta=u(11);
psi=u(12);

tgt_state=u(13:21);
time=u(22);

accel_in=u(23:25);
```

```
% ******  initialize variables  ******

% ************  functions  ************
if (Vc==0)
   tgo=1e6;
else
   tgo=R/Vc;
end
% compute relative state estimate
xhat=[R*cos(los);
   R*sin(los);
   R*sin(philos);
   tgt_state(2)-Vm*cos(psi);
   tgt_state(5)-Vm*sin(psi);
   tgt_state(8)-Vm*sin(theta);
   tgt_state(3);
   tgt_state(6);
   tgt_state(9);
   accel_in(1);
   accel_in(2);
   accel_in(3)];

if time<2.0
   ny=Nprime*Vc*(thetadot)/cos(heading-los);
   nz=Nprimez*Vc*(phidot)-9.8045;
else
   uc=(5/tgo^2)*[eye(3),tgo*eye(3),tgo^2/2*eye(3),zeros(3)]*xhat;
   ny=uc(2);
   nz=uc(3)-9.8045;

end

if satflag
   if (abs(ny)>30*9.8045)
      ny=sign(ny)*30*9.8045;
   end
   if (abs(nz)>30*9.8045)
      nz=sign(nz)*30*9.8045;
   end
end



Fx=0;
Fy=ny*m;
Fz=nz*m;

y=[ny;nz;Fx;Fy;Fz];

%//end of file chingfanlin.m
```

```
function y=draginduced(u)
%DRAGINDUCED Computes  induced aerodynamic drag force
%
%        see also
%        Copyright 1999-2000 by Triple B Enterprises
%//*********************************************************************
*********
%// File:  draginduced.m
%// Name:  LCDR Robert D. Broadston
%// MSEE/EE Thesis
%// Operating Environment:  Windows NT 4.0 Service Pack 5
%// Compiler:  MatLab v5.3
%// Date:  19 Sep 00
%// Description:  computes induced drag for simplified 6DOF
%// Inputs:  force output of guidance law, state
%// Outputs:  drag force
%// Process:  work backwards to CN from forces
%// Assumptions:
%// Warnings:
%//*********************************************************************
*********
%// Order of elements
%//      -Define globals
%//      -Define constants
%//      -Define elements of input vector
%//      -Functions
%//*********************************************************************
*********
% ******  define globals  ******
global SREF m

% ******  define constants  ******
eAR=1.5;                                % elliptical eff & AR

% ******  define input vector  ******
Fy=u(2);                                % y force
Fz=u(3);                                % z force
v2=u(7)^2+u(8)^2+u(9)^2;                % missile velocity
alt=u(6);                               % missile alt

% ******  initialize variables  ******
rho=rhovalt(abs(alt));                      % atmospheric density
mach=sqrt(v2)/machvalt(alt);
Q=rho*v2/2;                                 % dynamic pressure
```

```
% ************  functions  ************
if (Q==0)
   Cny=0;
   Cnz=0;
else
   Cny=Fy/(Q*SREF);                          % y normal coefficient
   Cnz=Fz/(Q*SREF);                          % z normal coefficient
end

Cdi=(Cny^2+Cnz^2)/(pi*eAR);                % induced drag coefficient

if (mach<1)                                % subsonic drag equal to
   Cdi=.25*sqrt(Fy^2+Fz^2)/(m*9.8045);     % max Cd0*applied G force
end

y=Cdi*Q*SREF;                              % drag force

%//end of file draginduced.m
```

```
function y=draginducedtbm(u)
%DRAGINDUCEDTBM Computes  induced aerodynamic drag force
%
%        see also
%        Copyright 1999-2000 by Triple B Enterprises
%//*********************************************************************
*********
%// File:  draginducedtbm.m
%// Name:  LCDR Robert D. Broadston
%// MSEE/EE Thesis
%// Operating Environment:  Windows NT 4.0 Service Pack 5
%// Compiler:  MatLab v5.3
%// Date:  19 Sep 00
%// Description:  computes induced drag for TBMD simulation
%// Inputs:  force output of guidance law, state
%// Outputs:  drag force
%// Process:  work backwards to CN from forces, corrects for ECI
%// Assumptions:
%// Warnings:
%//*********************************************************************
*********
%// Order of elements
%//      -Define globals
%//      -Define constants
%//      -Define elements of input vector
%//      -Functions
%//*********************************************************************
*********
% ******  define globals  ******
global SREF m

% ******  define constants  ******
eAR=1.5;                                % elliptical eff & AR

% ******  define input vector  ******
Fy=u(2);                                % y force
Fz=u(3);                                % z force
v2=u(7)^2+u(8)^2+u(9)^2;                % missile velocity
alt=sqrt(u(1)^2+u(2)^2+u(3)^2)-6371e3;  % missile alt

% ******  initialize variables  ******
rho=rhovalt(abs(alt));                      % atmospheric density
mach=sqrt(v2)/machvalt(alt);
Q=rho*v2/2;                                 % dynamic pressure
```

```
% ************  functions  ************
if (Q==0)
   Cny=0;
   Cnz=0;
else
   Cny=Fy/(Q*SREF);                          % y normal coefficient
   Cnz=Fz/(Q*SREF);                          % z normal coefficient
end

Cdi=(Cny^2+Cnz^2)/(pi*eAR);                  % induced drag coefficient

if (mach<1)                                  % subsonic drag equal to
   Cdi=.25*sqrt(Fy^2+Fz^2)/(m*9.8045);       % max Cd0*applied G force
end

y=Cdi*Q*SREF;                                % drag force

%//end of file draginducedtbm.m
```

```
function y=dragthesis(u)
%DRAGTHESIS Computes aerodynamic drag force
%
%       see also
%       Copyright 1999-2000 by Triple B Enterprises
%//*****************************************************************
*********
%// File:  dragthesis.m
%// Name:  LCDR Robert D. Broadston
%// MSEE/EE Thesis
%// Operating Environment:  Windows NT 4.0 Service Pack 5
%// Compiler:  MatLab v5.3
%// Date:  19 Sep 00
%// Description:  computes parasitic drag after breaking apart
%//               state vector
%// Inputs:  state vector, boost status
%// Outputs:  parasitic drag force
%// Process:
%// Assumptions:
%// Warnings:
%//*****************************************************************
*********
%// Order of elements
%//      -Define globals
%//      -Define constants
%//      -Define elements of input vector
%//      -Functions
%//*****************************************************************
*********
% ******  define globals  ******
global SREF

% ******  define constants  ******

% ******  define input vector  ******
vel2=u(4)^2+u(5)^2+u(6)^2;
alt=u(3);

boost=u(14);

% ******  initialize variables  ******

% ************  functions  ************
y=formdrag(SREF,alt,vel2,boost);

%//end of file dragthesis.m
```

```
%//*****************************************************************
*********
%// File:  drawmissile.m
%// Name:  LCDR Robert D. Broadston
%// MSEE/EE Thesis
%// Operating Environment:  Windows NT 4.0 Service Pack 5
%// Compiler:  MatLab v5.3
%// Date:  31 May 00
%// Description:  draws picture of current missile defined
%//               by missiledata#.dat
%// Inputs:
%// Outputs:
%// Process:
%// Assumptions:
%// Warnings:
%//*****************************************************************
*********
%// Order of elements
%//      -Define globals
%//      -Define constants
%//      -Define elements of input vector
%//      -Functions
%//*****************************************************************
*********
% ******  define globals  ******

% ******  define constants  ******
mgrey=[.5 .5 .5];
dgrey=[.75 .75 .75];

% ******  define input vector  ******

% ******  initialize variables  ******
nosex=[0 LN LN 0];
nosey=[0 d/2 -d/2 0];

bodyx=[LN L L LN LN];
bodyy=[d/2 d/2 -d/2 -d/2 d/2];

wingx=[LN+XW LN+XW+CRW LN+XW+CRW LN+XW+CRW-CTW LN+XW LN+XW];
wingy=[d/2 d/2 d/2+WXT+HW d/2+WXT+HW d/2+WXT d/2];

tailx=[L-CRT L L L-CTT L-CRT L-CRT];
taily=[d/2 d/2 d/2+TXT+HT d/2+TXT+HT d/2+TXT d/2];

CPEFF=(XCPN*AN+XCPB*AB+XCPW*SW+XHL*ST)/(AN+AB+SW+ST);

% compute time
time=rem(now,1);
hr=floor(time*24);
mins=floor(rem(time*24,1)*60);
timestr=[' ',num2str(hr),':',num2str(mins)];
```

```
% ************  functions  ************
figure(10)
clf
hold on
axis equal
fill(nosex,nosey,'w')
fill(bodyx,bodyy,mgrey)
fill(wingx,wingy,dgrey)
fill(wingx,-wingy,dgrey)
fill(tailx,taily,dgrey)
fill(tailx,-taily,dgrey)
plot(XCG,0,'ko')
plot(XHL,0,'r*')
plot(CPEFF,0,'b*')
legend('Center of Gravity','Hinge Line','Effective Center of Pressure')
title(['Missile Plan View      ',date,timestr])
xlabel('meters')
ylabel('meters')
hold off

%//end of file drawmissile.m
```

```
function y=dynamic3D(u)
%DYNAMIC3D Computes motion dymanics for a
%          body in three dimensions
%       see also
%       Copyright 1999-2000 by Triple B Enterprises
%//*****************************************************************
*********
%// File:  dynamic3d.m
%// Name:  LCDR Robert D. Broadston
%// MSEE/EE Thesis
%// Operating Environment:  Windows NT 4.0 Service Pack 5
%// Compiler:  MatLab v5.3
%// Date:  16 Feb 00
%// Description:  target motion dynamics
%// Inputs:  target state, turn rate input
%// Outputs:  derivative of target state
%// Process:
%// Assumptions:
%// Warnings:
%//*****************************************************************
*********
%// Order of elements
%//      -Define globals
%//      -Define constants
%//      -Define elements of input vector
%//      -Functions
%//*****************************************************************
*********
% ******  define globals  ******

% ******  define constants  ******

% ******  define input vector  ******
omega=u(1);
x=u(2);
xdot=u(3);
y=u(4);
ydot=u(5);
z=u(6);
zdot=u(7);

% ******  initialize variables  ******

% ************  functions  ************
y=[xdot;
   -omega*ydot;
   ydot;
   omega*xdot
   zdot;
   0];

%//end of file dynamic3d.m
```

```
function y=eqnforce(u)
%EQNFORCE Computes force dymanics for six degrees
%         of freedom flat earth model
%        see also
%        Copyright 1999-2000 by Triple B Enterprises
%//*********************************************************************
*********
%// File:  eqnforce.m
%// Name:  LCDR Robert D. Broadston
%// MSEE/EE Thesis
%// Operating Environment:  Windows NT 4.0 Service Pack 5
%// Compiler:  MatLab v5.3
%// Date:  10 May 00
%// Description:  force dynamics for full aero model
%// Inputs:  see below
%// Outputs:  solution to force equation
%// Process:
%// Assumptions:
%// Warnings:
%//*********************************************************************
*********
%// Order of elements
%//      -Define globals
%//      -Define constants
%//      -Define elements of input vector
%//      -Functions
%//*********************************************************************
*********
% ******  define globals  ******

% ******  define constants  ******

% ******  define input vector  ******
v_b=[u(1);u(2);u(3)];
F_B=[u(4);u(5);u(6)];
omega_b=[u(7);u(8);u(9)];
P=u(7);  Q=u(8);  R=u(9);

q=[u(10);u(11);u(12);u(13)];

magq=sqrt(q(1)^2+q(2)^2+q(3)^2+q(4)^2);
q=q/magq;

% the ever lovin' force of gravity
% g=[u(14);u(15);u(16)];
% note we are not using an external gravity model here

g=[0;0;9.8045];

% and lest we forget, mass
m=u(17);
```

```
% ******  initialize variables  ******

% ************  functions  ************
% some heavy duty number crunching
% compute rotation matrices
B=quat2b(q);

OMEGA_B=[0 -R  Q;
         R  0 -P;
        -Q  P  0];


y=-1*OMEGA_B*v_b+B*g+(1/m)*F_B;

%//end of file eqnforce.m
```

```
function y=eqnmoment(u)
%EQNMOMENT Computes moment dymanics for six degrees
%          of freedom  flat earth model
%        see also
%        Copyright 1999-2000 by Triple B Enterprises
%//*********************************************************************
*********
%// File:  eqnmoment.m
%// Name:  LCDR Robert D. Broadston
%// MSEE/EE Thesis
%// Operating Environment:  Windows NT 4.0 Service Pack 5
%// Compiler:  MatLab v5.3
%// Date:  11 Sep 00
%// Description:  computes moment dynamics for full aero model
%// Inputs:  see below
%// Outputs:  see below
%// Process:
%// Assumptions:
%// Warnings:
%//*********************************************************************
*********
%// Order of elements
%//      -Define globals
%//      -Define constants
%//      -Define elements of input vector
%//      -Functions
%//*********************************************************************
*********
% ******  define globals  ******
% ******  define constants  ******
% ******  define input vector  ******
omega_b=[u(1);u(2);u(3)];
P=u(1);  Q=u(2);  R=u(3);

% torques
T_B=[u(4);u(5);u(6)];

% inertial matrix
J=[u(7),0,0;
   0,u(8),0;
   0,0,u(9)];

% and lest we forget, mass
m=u(10);

% ******  initialize variables  ******
% ************  functions  ************
% some heavy duty number crunching
OMEGA_B=[0 -R  Q;
         R  0 -P;
        -Q  P  0];

y=-1*inv(J)*OMEGA_B*J*omega_b+inv(J)*T_B;

%//end of file eqnmoment.m
```

```
function y=eqnposit(u)
%EQNPOSIT Computes NED dymanics for six degrees
%         of freedom flat earth model
%        see also
%        Copyright 1999-2000 by Triple B Enterprises
%//*****************************************************************
*********
%// File:  eqnposit.m
%// Name:  LCDR Robert D. Broadston
%// MSEE/EE Thesis
%// Operating Environment:  Windows NT 4.0 Service Pack 5
%// Compiler:  MatLab v5.3
%// Date:  10 May 00
%// Description:  navigation equation for full aero model
%// Inputs:  see below
%// Outputs:  inertial velocities
%// Process:
%// Assumptions:
%// Warnings:
%//*****************************************************************
*********
%// Order of elements
%//      -Define globals
%//      -Define constants
%//      -Define elements of input vector
%//      -Functions
%//*****************************************************************
*********
% ******  define globals  ******

% ******  define constants  ******

% ******  define input vector  ******
v_b=[u(1);u(2);u(3)];

q=[u(4);u(5);u(6);u(7)];

% ******  initialize variables  ******
magq=sqrt(q(1)^2+q(2)^2+q(3)^2+q(4)^2);
q=q/magq;

% ************  functions  ************
% compute rotation matrices
B=quat2b(q);

y=B'*v_b;
%//end of file eqnposit.m
```

```
function y=eqnquat(u)
%EQNQUAT Computes quaternion dymanics for six degrees
%        of freedom flat earth model
%        see also
%        Copyright 1999-2000 by Triple B Enterprises
%//*********************************************************************
*********
%// File:  eqnquat.m
%// Name:  LCDR Robert D. Broadston
%// MSEE/EE Thesis
%// Operating Environment:  Windows NT 4.0 Service Pack 5
%// Compiler:  MatLab v5.3
%// Date:  13 Sep 00
%// Description:  computes quaternion dynamics for full aero model
%// Inputs:  see below
%// Outputs:  q_dot
%// Process:
%// Assumptions:
%// Warnings:
%//*********************************************************************
*********
%// Order of elements
%//      -Define globals
%//      -Define constants
%//      -Define elements of input vector
%//      -Functions
%//*********************************************************************
*********
% ******  define globals  ******
% ******  define constants  ******
% ******  define input vector  ******
q=[u(1);u(2);u(3);u(4)];

omega_b=[u(5);u(6);u(7)];
P=u(5);  Q=u(6);  R=u(7);

% ******  initialize variables  ******
magq=sqrt(q(1)^2+q(2)^2+q(3)^2+q(4)^2);
q=q/magq;

% ************  functions  ************
OMEGA_q=[0  P  Q  R;
        -P  0 -R  Q;
        -Q  R  0 -P;
        -R -Q  P  0];

q=-(1/2)*OMEGA_q*q;
magq=sqrt(q(1)^2+q(2)^2+q(3)^2+q(4)^2);
if (magq~=0)
   y=q/magq;
else
   y=[1;0;0;0];
end

%//end of file eqnquat.m
```

```
function y=flatearthdyn(u)
%FLATEARTHDYN Computes motion dymanics for six degrees
%          of freedom for a flat earth model
%        see also
%        Copyright 1999-2000 by Triple B Enterprises
%//*********************************************************************
*********
%// File:  flatearthdyn.m
%// Name:  LCDR Robert D. Broadston
%// MSEE/EE Thesis
%// Operating Environment:  Windows NT 4.0 Service Pack 5
%// Compiler:  MatLab v5.3
%// Date:  1 Aug 00
%// Description:  computes 6DOF dynamics for flat earth using
%//               quaternion formulation
%// Inputs:  see below
%// Outputs:  derivative of state vector
%// Process:  Stevens & Lewis
%// Assumptions:
%// Warnings:
%//*********************************************************************
*********
%// Order of elements
%//      -Define globals
%//      -Define constants
%//      -Define elements of input vector
%//      -Functions
%//*********************************************************************
*********
% ******  define globals  ******

% ******  define constants  ******

% ******  define input vector  ******
p=[u(1);u(2);u(3)];
v_b=[u(4);u(5);u(6)];
omega_b=[u(7);u(8);u(9)];
P=u(7);  Q=u(8);  R=u(9);

q=[u(10);u(11);u(12);u(13)];

magq=sqrt(q(1)^2+q(2)^2+q(3)^2+q(4)^2);
q=q/magq;

x=[p;v_b;omega_b;q];

% inertial matrix
J=[u(14),0,0;
   0,u(15),0;
   0,0,u(16)];


% forces
F_B=[u(17);u(18);u(19)];
```

```
% torques
T_B=[u(20);u(21);u(22)];

% the ever lovin' force of gravity
% note we are not using an external gravity model here

g=[0;0;9.8045];

% and lest we forget, mass
m=u(26);

% ******  initialize variables  ******
% compute rotation matrices
B=quat2b(q);

OMEGA_B=[0 -R  Q;
         R  0 -P;
        -Q  P  0];

OMEGA_q=[0  P  Q  R;
        -P  0 -R  Q;
        -Q  R  0 -P;
        -R -Q  P  0];

% ************  functions  ************
y=[   zeros(3),        B',                    zeros(3),      zeros(3,4);
      zeros(3),    -OMEGA_B,                  zeros(3),      zeros(3,4);
   zeros(3),     zeros(3),           -1*inv(J)*OMEGA_B*J, zeros(3,4);
   zeros(4,3),   zeros(4,3),             zeros(4,3),   (-1/2)*OMEGA_q];
y=y*x;

y=y+[zeros(3,1);
   B*g+(1/m)*F_B;
   inv(J)*T_B;
   zeros(4,1)];

%//end of file flatearthdyn.m
```

```
function y=formdrag(A,alt,vel2,boost)
%FORMDRAG Computes form drag for a missile with frontal
%        area A in a standard atmosphere
%        FORMDRAG(A,ALT,VEL2,BOOST)
%        uses MACHVALT,CDVMACH,RHOVALT
%        see also
%        Copyright 1999-2000 by Triple B Enterprises
%//*****************************************************************
*********
%// File:  formdrag.m
%// Name:  LCDR Robert D. Broadston
%// MSEE/EE Thesis
%// Operating Environment:  Windows NT 4.0 Service Pack 5
%// Compiler:  MatLab v5.3
%// Date:  9 May 00
%// Description:  Computes form drag for a missile with frontal
%                 area A in a standard atmosphere
%// Inputs:  area, altitude, V^2, boost on/off
%// Outputs:  parasitic drag force
%// Process:
%// Assumptions:
%// Warnings:
%//*****************************************************************
*********
%// Order of elements
%//      -Define globals
%//      -Define constants
%//      -Define elements of input vector
%//      -Functions
%//*****************************************************************
*********
% ******  define globals  ******

% ******  define constants  ******

% ******  define input vector  ******

% ******  initialize variables  ******
rho=rhovalt(alt);
mach=(vel2)^(1/2)/machvalt(alt);

% ************  functions  ************
if (mach>100)
   mach=.83;
end

Cd=cdvmach(mach,boost);

y=rho*vel2*Cd*A/2;

%//end of file formdrag.m
```

```
function y=gravity(u)
%GRAVITY Computes simple gravity model for 6DOF model
%        see also
%        Copyright 1999-2000 by Triple B Enterprises
%//*********************************************************************
*********
%// File:  gravity.m
%// Name:  LCDR Robert D. Broadston
%// MSEE/EE Thesis
%// Operating Environment:  Windows NT 4.0 Service Pack 5
%// Compiler:  MatLab v5.3
%// Date:  19 Sep 00
%// Description:  computes spherical earth gravity for TBM
%//               target dynamics
%// Inputs:  target state vector
%// Outputs:  gravity vector
%// Process:
%// Assumptions:
%// Warnings:
%//*********************************************************************
*********
%// Order of elements
%//      -Define globals
%//      -Define constants
%//      -Define elements of input vector
%//      -Functions
%//*********************************************************************
*********
% ******  define globals  ******

% ******  define constants  ******
GM_E=3.9860014e14;       % G*mass of earth

% ******  define input vector  ******

% ******  initialize variables  ******
mag=sqrt(u(1)^2+u(3)^2+u(5)^2);

% ************  functions  ************
y=-(GM_E/mag^3)*[u(1);u(3);u(5)];

%//end of file gravity.m
```

```
function y=gravity2(u)
%GRAVITY2 Computes simple gravity model for 6DOF model
%        see also
%        Copyright 1999-2000 by Triple B Enterprises
%//*********************************************************************
*********
%// File:  gravity2.m
%// Name:  LCDR Robert D. Broadston
%// MSEE/EE Thesis
%// Operating Environment:  Windows NT 4.0 Service Pack 5
%// Compiler:  MatLab v5.3
%// Date:  19 Sep 00
%// Description:  computes spherical earth gravity model
%//               for interceptor
%// Inputs:  missile state vector
%// Outputs:  gravity vector
%// Process:
%// Assumptions:
%// Warnings:
%//*********************************************************************
*********
%// Order of elements
%//      -Define globals
%//      -Define constants
%//      -Define elements of input vector
%//      -Functions
%//*********************************************************************
*********
% ******  define globals  ******

% ******  define constants  ******
GM_E=3.9860014e14;       % G*mass of earth

% ******  define input vector  ******

% ******  initialize variables  ******
mag=sqrt(u(1)^2+u(2)^2+u(3)^2);

% ************  functions  ************
y=-(GM_E/mag^3)*[u(1);u(2);u(3)];

%//end of file gravity2.m
```

```
%//*************************************************************************
*********
%// File:  KBFILTER.m
%// Name:  LCDR Robert D. Broadston
%// MSEE/EE Thesis
%// Operating Environment:  Windows NT 4.0 Service Pack 5
%// Compiler:  MatLab v5.3
%// Date:  7 Aug 00
%// Description:  Automatically computes a kinematic boundary using
6DOF
%//               simulator with tracking filter.
%//               -Streamlined search loops
%//               -Status indicator
%//               -Saves most recent data to disk
%//               -Derived from KBOUTER
%// Inputs:  none
%// Outputs:  one figure of kinematic boundary
%// Process:  streamlined brute force search algorithm
%// Assumptions:  none
%// Warnings:  none
%//*************************************************************************
*********
%// Order of elements
%//      -Define globals
%//      -Define constants
%//      -Define elements of input vector
%//      -Initialize variables
%//      -Functions
%//*************************************************************************
*********

% ******  define globals  ******

% ******  define constants  ******
thesisinit
% set min engagement range (10000 m default)
minrng=10000;
% set heading increment
degstep=5;

% ******  define input vector  ******

% ******  initialize variables  ******

maxhit=[];   minhit=[];
load current
% set target altitude
tgtalt=init(3);                 % default co-altitude
% set target turn rate.  default=0 degrees/sec
target_turn=0;
% set target speed
tgtmach=.83;                    % user sets Mach #
tgtspd=tgtmach*machvalt(tgtalt);% machine computes speed
```

```
% ************  functions  ************
% start in tail chase step to head on by <degstep>
% degree increments
for heading=0:degstep:180
   tic
   plotcount=1;  runplot=[]; rangemax=0;
   heading                    % show heading counter
   tgthdg=heading*pi/180;

   % compute target speed components
   xspd=tgtspd*cos(tgthdg);
   yspd=tgtspd*sin(tgthdg);

   % first range loop step by 10 km
   for tgtrng=minrng:10000:150000
      disp(['*** ',num2str(tgtrng),' ***'])
      % set initial target state
      tgtinit=[tgtrng;xspd;0;yspd;tgtalt;0];
      % initialize filter

XLAST=[tgtinit(1);tgtinit(2);0;tgtinit(3);tgtinit(4);0;tgtinit(5);tgtin
it(6);0];

      % call simulation
      sim('thesis1filt')

      % analyze data from current run
      range=sqrt((MissileOut(:,1)-TgtOut(:,1)).^2+...
         (MissileOut(:,2)-TgtOut(:,3)).^2+...
         (MissileOut(:,3)-TgtOut(:,5)).^2);
      % save run data
      runplot(plotcount,:)=[min(range),tgtrng];
      % score run
      if (min(range)>5)
         break
      end
      plotcount=plotcount+1;
   end

   idx=find(runplot(:,1)<=5);
   if(idx)
      rangemax=runplot(max(idx),2);
   end
   runplot=[];
   plotcount=1;
```

```
   % 1 km step size for max range.  Streamlining code
   tgtrng=rangemax+1000;
   disp(['*** ',num2str(tgtrng),' max1k***'])
   % set initial target state
   tgtinit=[tgtrng;xspd;0;yspd;tgtalt;0];
   % initialize filter

XLAST=[tgtinit(1);tgtinit(2);0;tgtinit(3);tgtinit(4);0;tgtinit(5);tgtin
it(6);0];

   % call simulation
   sim('thesis1filt')

   % analyze data from current run
   range=sqrt((MissileOut(:,1)-TgtOut(:,1)).^2+...
      (MissileOut(:,2)-TgtOut(:,3)).^2+...
      (MissileOut(:,3)-TgtOut(:,5)).^2);
   if (min(range)<=5)
      rangemax=tgtrng;
      tgtrng=tgtrng+4000;
      disp(['*** ',num2str(tgtrng),' max1k***'])
      % set initial target state
      tgtinit=[tgtrng;xspd;0;yspd;tgtalt;0];
      % initialize filter

XLAST=[tgtinit(1);tgtinit(2);0;tgtinit(3);tgtinit(4);0;tgtinit(5);tgtin
it(6);0];

      % call simulation
      sim('thesis1filt')

      % analyze data from current run
      range=sqrt((MissileOut(:,1)-TgtOut(:,1)).^2+...
         (MissileOut(:,2)-TgtOut(:,3)).^2+...
         (MissileOut(:,3)-TgtOut(:,5)).^2);
      if (min(range)<=5)
         rangemax=tgtrng;
      end
   end
   % main search loop 1km step size
   for tgtrng=rangemax+1000:1000:(rangemax+4000)

      disp(['*** ',num2str(tgtrng),' max1k***'])
      % set initial target state
      tgtinit=[tgtrng;xspd;0;yspd;tgtalt;0];
      % initialize filter

XLAST=[tgtinit(1);tgtinit(2);0;tgtinit(3);tgtinit(4);0;tgtinit(5);tgtin
it(6);0];

      % call simulation
      sim('thesis1filt')
```

```
      % analyze data from current run
      range=sqrt((MissileOut(:,1)-TgtOut(:,1)).^2+...
         (MissileOut(:,2)-TgtOut(:,3)).^2+...
         (MissileOut(:,3)-TgtOut(:,5)).^2);

      % save run data
      runplot(plotcount,:)=[min(range),tgtrng];
      % score run
      if (min(range)>5)
         break
      end

      plotcount=plotcount+1;
   end

   idx=find(runplot(:,1)<=5);
   if(idx)
      rangemax=runplot(max(idx),2);
   end
   runplot=[];
   plotcount=1;

% 100 m step size for max range.  Streamlined code.
   tgtrng=rangemax+100;
   disp(['*** ',num2str(tgtrng),' max100***'])
   % set initial target state
   tgtinit=[tgtrng;xspd;0;yspd;tgtalt;0];
   % initialize filter

XLAST=[tgtinit(1);tgtinit(2);0;tgtinit(3);tgtinit(4);0;tgtinit(5);tgtin
it(6);0];

   % call simulation
   sim('thesis1filt')

   % analyze data from current run
   range=sqrt((MissileOut(:,1)-TgtOut(:,1)).^2+...
      (MissileOut(:,2)-TgtOut(:,3)).^2+...
      (MissileOut(:,3)-TgtOut(:,5)).^2);
   if (min(range)<=5)
      rangemax=tgtrng;
      tgtrng=tgtrng+400;
      disp(['*** ',num2str(tgtrng),' max100***'])
      % set initial target state
      tgtinit=[tgtrng;xspd;0;yspd;tgtalt;0];
      % initialize filter

XLAST=[tgtinit(1);tgtinit(2);0;tgtinit(3);tgtinit(4);0;tgtinit(5);tgtin
it(6);0];

      % call simulation
      sim('thesis1filt')
```

```
      % analyze data from current run
      range=sqrt((MissileOut(:,1)-TgtOut(:,1)).^2+...
         (MissileOut(:,2)-TgtOut(:,3)).^2+...
         (MissileOut(:,3)-TgtOut(:,5)).^2);
      if (min(range)<=5)
         rangemax=tgtrng;
      end
   end

   % main search loop 100 m
   for tgtrng=rangemax+100:100:(rangemax+400)

      disp(['*** ',num2str(tgtrng),' max100***'])
      % set initial target state
      tgtinit=[tgtrng;xspd;0;yspd;tgtalt;0];
      % initialize filter

XLAST=[tgtinit(1);tgtinit(2);0;tgtinit(3);tgtinit(4);0;tgtinit(5);tgtin
it(6);0];

      % call simulation
      sim('thesis1filt')

      % analyze data from current run
      range=sqrt((MissileOut(:,1)-TgtOut(:,1)).^2+...
         (MissileOut(:,2)-TgtOut(:,3)).^2+...
         (MissileOut(:,3)-TgtOut(:,5)).^2);

      % save run data
      runplot(plotcount,:)=[min(range),tgtrng];
      % score run
      if (min(range)>5)
         break
      end

      plotcount=plotcount+1;
   end

   idx=find(runplot(:,1)<=5);
   if(idx)
      rangemax=runplot(max(idx),2);
   end
   runplot=[];
   plotcount=1;
```

```
   % 10 m step size for max range.  Streamlined code
   tgtrng=rangemax+10;
   disp(['*** ',num2str(tgtrng),' max10***'])
   % set initial target state
   tgtinit=[tgtrng;xspd;0;yspd;tgtalt;0];
   % initialize filter

XLAST=[tgtinit(1);tgtinit(2);0;tgtinit(3);tgtinit(4);0;tgtinit(5);tgtin
it(6);0];

   % call simulation
   sim('thesis1filt')

   % analyze data from current run
   range=sqrt((MissileOut(:,1)-TgtOut(:,1)).^2+...
      (MissileOut(:,2)-TgtOut(:,3)).^2+...
      (MissileOut(:,3)-TgtOut(:,5)).^2);
   if (min(range)<=5)
      rangemax=tgtrng;
      tgtrng=tgtrng+40;
      disp(['*** ',num2str(tgtrng),' max10***'])
      % set initial target state
      tgtinit=[tgtrng;xspd;0;yspd;tgtalt;0];
      % initialize filter

XLAST=[tgtinit(1);tgtinit(2);0;tgtinit(3);tgtinit(4);0;tgtinit(5);tgtin
it(6);0];

      % call simulation
      sim('thesis1filt')

      % analyze data from current run
      range=sqrt((MissileOut(:,1)-TgtOut(:,1)).^2+...
         (MissileOut(:,2)-TgtOut(:,3)).^2+...
         (MissileOut(:,3)-TgtOut(:,5)).^2);
      if (min(range)<=5)
         rangemax=tgtrng;
      end
   end

   % main search loop 10 m.  Note, we are now computing the
   % full output vector for each run.
   for tgtrng=rangemax:10:(rangemax+40)

      disp(['*** ',num2str(tgtrng),' max10***'])
      % set initial target state
      tgtinit=[tgtrng;xspd;0;yspd;tgtalt;0];
      % initialize filter

XLAST=[tgtinit(1);tgtinit(2);0;tgtinit(3);tgtinit(4);0;tgtinit(5);tgtin
it(6);0];

      % call simulation
      sim('thesis1filt')
```

```
        % analyze data from current run
        range=sqrt((MissileOut(:,1)-TgtOut(:,1)).^2+...
           (MissileOut(:,2)-TgtOut(:,3)).^2+...
           (MissileOut(:,3)-TgtOut(:,5)).^2);
           t=MissileOut(:,14);
        index=find(range==min(range));
        ip=t(index(1));

        % compute cost function J=20*e(tf)^2+integ(u^2)/200
        % and missile divert
        u2=(omegaout(:,1).^2+omegaout(:,2).^2);
        integral=0;
        for ii=2:index
           integral=integral+(t(ii)-t(ii-1))*u2(ii-1);
        end
        J=20*min(range)^2+integral/1000;

        % save run data [miss dist,cost,divert,time,max range]
        runplot(plotcount,:)=[min(range),J,integral,ip,tgtrng];

        if (min(range)>5)
           break
        end

        plotcount=plotcount+1;
     end

     idx=find(runplot(:,1)<=5);
     if(idx)
        rangemax=runplot(max(idx),5);
     end

     if (isempty(idx))
        maxhit(heading+1,:)=[0,0,0,0,0];
     else
        maxhit(heading+1,:)=runplot(max(idx),:);
     end

     runplot=[];
     plotcount=1;

     % save data to disk
     save current maxhit
     toc
     % note for some guidance laws, the down step here
     % must be 2 or more-----------------|
     minrng=10000*(floor(rangemax/10000)-1);
     if (minrng<=5000)
        minrng=10000;
     end

end
```

```
% plot it for me baby
rho1=maxhit(1:degstep:181,5);
rho1=[rho1;flipud(rho1)];
theta=180:degstep:360;
theta=pi/180*theta;
theta=[theta,-1*fliplr(theta)]';
figure(5)
polar(theta,rho1)

%//end of file KBOUTER.m
```

```
%//*************************************************************************
*********
%// File:  KBOUTER2.m
%// Name:  LCDR Robert D. Broadston
%// MSEE/EE Thesis
%// Operating Environment:  Windows NT 4.0 Service Pack 5
%// Compiler:  MatLab v5.3
%// Date:  7 Aug 00
%// Description:  Automatically computes a kinematic boundary using
6DOF
%//               simulator.
%//               -Streamlined search loops
%//               -Status indicator
%//               -Saves most recent data to disk
%//               -Derived from KBOUTER
%// Inputs:  none
%// Outputs:  one figure of kinematic boundary
%// Process:  streamlined brute force search algorithm
%// Assumptions:  none
%// Warnings:  none
%//*************************************************************************
*********
%// Order of elements
%//      -Define globals
%//      -Define constants
%//      -Define elements of input vector
%//      -Initialize variables
%//      -Functions
%//*************************************************************************
*********

% ******  define globals  ******

% ******  define constants  ******
thesisinit
% set target turn rate, default=0
target_turn=0;
% set min engagement range (10000 m default)
minrng=10000;
% set heading increment
degstep=5;

% ******  define input vector  ******

% ******  initialize variables  ******

maxhit=[];   minhit=[];
load current
% set target altitude
tgtalt=50;%init(3);                % default co-altitude

% set target speed
tgtmach=.83;                    % user sets Mach #
tgtspd=tgtmach*machvalt(tgtalt);% machine computes speed
```

```
% ************  functions  ************
% start in tail chase step to head on by <degstep>
% degree increments
for heading=0:degstep:180
   tic
   plotcount=1;  runplot=[]; rangemax=0;
   heading                      % show heading counter
   tgthdg=heading*pi/180;

   % compute target speed components
   xspd=tgtspd*cos(tgthdg);
   yspd=tgtspd*sin(tgthdg);

   % first range loop step by 10 km
   for tgtrng=minrng:10000:150000
      disp(['*** ',num2str(tgtrng),' ***'])
      % set initial target state
      tgtinit=[tgtrng;xspd;0;yspd;tgtalt;0];

      % call simulation
      sim('Thesis1')

      % analyze data from current run
      range=sqrt((MissileOut(:,1)-TgtOut(:,1)).^2+...
         (MissileOut(:,2)-TgtOut(:,3)).^2+...
         (MissileOut(:,3)-TgtOut(:,5)).^2);
      % save run data
      runplot(plotcount,:)=[min(range),tgtrng];
      % score run
      if (min(range)>5)
         break
      end
      plotcount=plotcount+1;
   end

   idx=find(runplot(:,1)<=5);
   if(idx)
      rangemax=runplot(max(idx),2);
   end
   runplot=[];
   plotcount=1;

   % 1 km step size for max range.  Streamlining code
   tgtrng=rangemax+1000;
   disp(['*** ',num2str(tgtrng),' max1k***'])
   % set initial target state
   tgtinit=[tgtrng;xspd;0;yspd;tgtalt;0];

   % call simulation
   sim('Thesis1')
```

```
% analyze data from current run
range=sqrt((MissileOut(:,1)-TgtOut(:,1)).^2+...
   (MissileOut(:,2)-TgtOut(:,3)).^2+...
   (MissileOut(:,3)-TgtOut(:,5)).^2);
if (min(range)<=5)
   rangemax=tgtrng;
   tgtrng=tgtrng+4000;
   disp(['*** ',num2str(tgtrng),' max1k***'])
   % set initial target state
   tgtinit=[tgtrng;xspd;0;yspd;tgtalt;0];


   % call simulation
   sim('Thesis1')

   % analyze data from current run
   range=sqrt((MissileOut(:,1)-TgtOut(:,1)).^2+...
      (MissileOut(:,2)-TgtOut(:,3)).^2+...
      (MissileOut(:,3)-TgtOut(:,5)).^2);
   if (min(range)<=5)
      rangemax=tgtrng;
   end
end

% main search loop 1km step size
for tgtrng=rangemax+1000:1000:(rangemax+4000)

   disp(['*** ',num2str(tgtrng),' max1k***'])
   % set initial target state
   tgtinit=[tgtrng;xspd;0;yspd;tgtalt;0];

   % call simulation
   sim('Thesis1')

   % analyze data from current run
   range=sqrt((MissileOut(:,1)-TgtOut(:,1)).^2+...
      (MissileOut(:,2)-TgtOut(:,3)).^2+...
      (MissileOut(:,3)-TgtOut(:,5)).^2);

   % save run data
   runplot(plotcount,:)=[min(range),tgtrng];
   % score run
   if (min(range)>5)
      break
   end

   plotcount=plotcount+1;
end

idx=find(runplot(:,1)<=5);
if(idx)
   rangemax=runplot(max(idx),2);
end
runplot=[];
plotcount=1;
```

```
% 100 m step size for max range.  Streamlined code.
tgtrng=rangemax+100;
disp(['*** ',num2str(tgtrng),' max100***'])
% set initial target state
tgtinit=[tgtrng;xspd;0;yspd;tgtalt;0];

% call simulation
sim('Thesis1')

% analyze data from current run
range=sqrt((MissileOut(:,1)-TgtOut(:,1)).^2+...
   (MissileOut(:,2)-TgtOut(:,3)).^2+...
   (MissileOut(:,3)-TgtOut(:,5)).^2);
if (min(range)<=5)
   rangemax=tgtrng;
   tgtrng=tgtrng+400;
   disp(['*** ',num2str(tgtrng),' max100***'])
   % set initial target state
   tgtinit=[tgtrng;xspd;0;yspd;tgtalt;0];


   % call simulation
   sim('Thesis1')

   % analyze data from current run
   range=sqrt((MissileOut(:,1)-TgtOut(:,1)).^2+...
      (MissileOut(:,2)-TgtOut(:,3)).^2+...
      (MissileOut(:,3)-TgtOut(:,5)).^2);
   if (min(range)<=5)
      rangemax=tgtrng;
   end
end

% main search loop 100 m
for tgtrng=rangemax+100:100:(rangemax+400)

   disp(['*** ',num2str(tgtrng),' max100***'])
   % set initial target state
   tgtinit=[tgtrng;xspd;0;yspd;tgtalt;0];


   % call simulation
   sim('Thesis1')

   % analyze data from current run
   range=sqrt((MissileOut(:,1)-TgtOut(:,1)).^2+...
      (MissileOut(:,2)-TgtOut(:,3)).^2+...
      (MissileOut(:,3)-TgtOut(:,5)).^2);
```

```
    % save run data
    runplot(plotcount,:)=[min(range),tgtrng];
    % score run
    if (min(range)>5)
       break
    end

    plotcount=plotcount+1;
end

idx=find(runplot(:,1)<=5);
if(idx)
   rangemax=runplot(max(idx),2);
end
runplot=[];
plotcount=1;

% 10 m step size for max range.  Streamlined code
tgtrng=rangemax+10;
disp(['*** ',num2str(tgtrng),' max10***'])
% set initial target state
tgtinit=[tgtrng;xspd;0;yspd;tgtalt;0];

% call simulation
sim('Thesis1')

% analyze data from current run
range=sqrt((MissileOut(:,1)-TgtOut(:,1)).^2+...
   (MissileOut(:,2)-TgtOut(:,3)).^2+...
   (MissileOut(:,3)-TgtOut(:,5)).^2);
if (min(range)<=5)
   rangemax=tgtrng;
   tgtrng=tgtrng+40;
   disp(['*** ',num2str(tgtrng),' max10***'])
   % set initial target state
   tgtinit=[tgtrng;xspd;0;yspd;tgtalt;0];


   % call simulation
   sim('Thesis1')

   % analyze data from current run
   range=sqrt((MissileOut(:,1)-TgtOut(:,1)).^2+...
      (MissileOut(:,2)-TgtOut(:,3)).^2+...
      (MissileOut(:,3)-TgtOut(:,5)).^2);
   if (min(range)<=5)
      rangemax=tgtrng;
   end
end
```

```
% main search loop 10 m.  Note, we are now computing the
% full output vector for each run.
for tgtrng=rangemax:10:(rangemax+40)

   disp(['*** ',num2str(tgtrng),' max10***'])
   % set initial target state
   tgtinit=[tgtrng;xspd;0;yspd;tgtalt;0];


   % call simulation
   sim('Thesis1')

   % analyze data from current run
   range=sqrt((MissileOut(:,1)-TgtOut(:,1)).^2+...
      (MissileOut(:,2)-TgtOut(:,3)).^2+...
      (MissileOut(:,3)-TgtOut(:,5)).^2);
      t=MissileOut(:,14);
   index=find(range==min(range));
   ip=t(index(1));

   % compute cost function J=20*e(tf)^2+integ(u^2)/200
   % and missile divert
   u2=(omegaout(:,1).^2+omegaout(:,2).^2);
   integral=0;
   for ii=2:index
      integral=integral+(t(ii)-t(ii-1))*u2(ii-1);
   end
   J=20*min(range)^2+integral/1000;

   % save run data [miss dist,cost,divert,time,max range]
   runplot(plotcount,:)=[min(range),J,integral,ip,tgtrng];

   if (min(range)>5)
      break
   end

   plotcount=plotcount+1;
end

idx=find(runplot(:,1)<=5);
if(idx)
   rangemax=runplot(max(idx),5);
end

if (isempty(idx))
   maxhit(heading+1,:)=[0,0,0,0,0];
else
   maxhit(heading+1,:)=runplot(max(idx),:);
end

runplot=[];
plotcount=1;
```

```
% save data to disk
   save current maxhit
   toc
   % note for some guidance laws, the down step here
   % must be 2 or more-----------------|
   minrng=10000*(floor(rangemax/10000)-1);
   if (minrng<=5000)
      minrng=5000;
   end

end

% plot it for me baby
rho1=maxhit(1:degstep:181,5);
rho1=[rho1;flipud(rho1)];
theta=180:degstep:360;
theta=pi/180*theta;
theta=[theta,-1*fliplr(theta)]';
figure(5)
polar(theta,rho1)

%//end of file KBOUTER.m
```

```
function y=machvalt(alt)
%MACHVALT   Computes linear approximation for a given
%           altitude in meters/sec based on standard ICAO
%           atmosphere
%        MACHVALT(ALT)
%        see also CDVMACH
%        Copyright 1999-2000 by Triple B Enterprises
%//*****************************************************************
*********
%// File:  machvalt.m
%// Name:  LCDR Robert D. Broadston
%// MSEE/EE Thesis
%// Operating Environment:  Windows NT 4.0 Service Pack 5
%// Compiler:  MatLab v5.3
%// Date:  8 Jun 00
%// Description:  computes linear approximation to Mach 1 for
%//               standard ICAO atmosphere
%// Inputs:  altitude
%// Outputs:  Mach 1
%// Process:
%// Assumptions:
%// Warnings:
%//*****************************************************************
*********
%// Order of elements
%//      -Define globals
%//      -Define constants
%//      -Define elements of input vector
%//      -Functions
%//*****************************************************************
*********
% ******  define globals  ******

% ******  define constants  ******
Mach1=[-.0041 340.3];
Mach2=295.1;
Mach3=[.00067 281.7];

% ******  define input vector  ******

% ******  initialize variables  ******
alt=abs(alt);                % account for NED coords

% ************  functions  ************
if (alt<11000)
   y=polyval(Mach1,alt);
else
   if (alt>20000)
      y=polyval(Mach3,alt);
   else
      y=Mach2;
   end
end

%//end of file machvalt.m
```

```
%//*********************************************************************
*********
%// File:  missiledata.m
%// Name:  LCDR Robert D. Broadston
%// MSEE/EE Thesis
%// Operating Environment:  Windows NT 4.0 Service Pack 5
%// Compiler:  MatLab v5.3
%// Date:  17 May 00
%// Description:  missile data for AMRAAM
%// Inputs:  none
%// Outputs:  various
%// Process:
%// Assumptions:
%// Warnings:
%//*********************************************************************
*********
%// Order of elements
%//      -Define globals
%//      -Define constants
%//      -Define elements of input vector
%//      -Functions
%//*********************************************************************
*********
% Establishes missile dimensions for use in computing
% Aerodynamic forces and moments
% Except where noted, all dimensions in MKS system

% Missile Name:  PSEUDO AMRAAM

% ******  define globals  ******
global m d L XCG XCPN XCPW XCPB XHL
global ST SW SPLAN SREF

% ******  define constants  ******
%------------ missile body dimensions -----------------------
m=156.8;                  % mass, may be time varying
d=.1778;                  % diameter
L=3.657;                   % length
XCG=1.8288;                 % initial c.g., may be time varying
LN=.6769;                  % length of nose cone
%------------ tailplane dimensions --------------------------
XHL=3.454;                 % hinge line arm
CRT=.4061;                 % tail root chord
CTT=.0676;                 % tail tip chord
TXT=.0676;                 % tail extension
HT=.2286;                  % tail height
%--------------- wing dimensions --------------------------
XW=1.134;                  % wing to radome tangency point
CRW=.3554;                 % wing root chord
CTW=0;                    % wing tip chord
WXT=0;                    % wing extension
HW=.1778;                 % wing height
```

```
% ******  define input vector  ******

% ******  initialize variables  ******

% ************  functions  ************
%-------------- centers of pressure -------------------------
XCPN=.67*LN;                              % nose CP
XCPW=LN+XW+.7*CRW-.2*CTW;                 % wing CP
AN=.67*LN*d;                              % plan area of nose
AB=(L-LN)*d;                              % plan area of body
XCPB=(.67*AN*LN+AB*(LN+.5*(L-LN)))/...    % body CP
   (AN+AB);
%-------------- area computation ----------------------------
SW=.5*HW*(CTW+CRW)+CRW*WXT;     % wing area
ST=.5*HT*(CTT+CRT)+CRT*TXT;     % tail area
SPLAN=(L-LN)*d+.67*L*d;         % body and nose plan area
SREF=pi*d^2/4;                  % missile cross section

%-------------- compute the inertial matrix ------------------
r=d/2;
Jx=m*r^2/2;
Jy=m*(L^2/12+r^2/4)+m*(L/2-XCG)^2;
Jz=Jy;

%//end of file missiledata.m
```

```
%//*********************************************************************
*********
%// File:  missiledata2.m
%// Name:  LCDR Robert D. Broadston
%// MSEE/EE Thesis
%// Operating Environment:  Windows NT 4.0 Service Pack 5
%// Compiler:  MatLab v5.3
%// Date:  12 Apr 00
%// Description:  missile data for Jerger missile from Zarchan
%// Inputs:  none
%// Outputs:  various
%// Process:
%// Assumptions:
%// Warnings:
%//*********************************************************************
*********
%// Order of elements
%//      -Define globals
%//      -Define constants
%//      -Define elements of input vector
%//      -Functions
%//*********************************************************************
*********
% Establishes missile dimensions for use in computing
% Aerodynamic forces and moments
% Except where noted, all dimensions in MKS system

% Missile Name:  JERGER

% ******  define globals  ******
global m d L XCG XCPN XCPW XCPB XHL
global ST SW SPLAN SREF

% ******  define constants  ******
%------------ missile body dimensions -----------------------
m=454.5444;                  % mass, may be time varying
d=.3048;                  % diameter
L=6.096;                   % length
XCG=3.048;                 % initial c.g., may be time varying
LN=.9144;                  % length of nose cone
%------------ tailplane dimensions --------------------------
XHL=5.9436;                 % hinge line arm
CRT=.6096;                 % tail root chord
CTT=.0;                 % tail tip chord
TXT=.0;                 % tail extension
HT=.6096;                   % tail height
%--------------- wing dimensions --------------------------
XW=1.2192;                   % wing to radome tangency point
CRW=1.8288;                  % wing root chord
CTW=0;                    % wing tip chord
WXT=0;                    % wing extension
HW=.6096;                 % wing height
```

```
% ******  define input vector  ******

% ******  initialize variables  ******

% ************  functions  ************
%--------------- centers of pressure -------------------------
XCPN=.67*LN;                              % nose CP
XCPW=LN+XW+.7*CRW-.2*CTW;                 % wing CP
AN=.67*LN*d;                              % plan area of nose
AB=(L-LN)*d;                              % plan area of body
XCPB=(.67*AN*LN+AB*(LN+.5*(L-LN)))/...    % body CP
   (AN+AB);
%--------------- area computation ----------------------------
SW=.5*HW*(CTW+CRW)+CRW*WXT;     % wing area
ST=.5*HT*(CTT+CRT)+CRT*TXT;     % tail area
SPLAN=(L-LN)*d+.67*L*d;         % body and nose plan area
SREF=pi*d^2/4;                  % missile cross section


%----------- compute the inertial matrix --------------------
r=d/2;
Jx=m*r^2/2;
Jy=m*(L^2/12+r^2/4);
Jz=Jy;

%//end of file missiledata2.m
```

```
%//*************************************************************************
*********
%// File:  missiledata3.m
%// Name:  LCDR Robert D. Broadston
%// MSEE/EE Thesis
%// Operating Environment:  Windows NT 4.0 Service Pack 5
%// Compiler:  MatLab v5.3
%// Date:  25 Aug 00
%// Description:  computes missile data for SM-2 MR
%// Inputs:  none
%// Outputs:  various
%// Process:
%// Assumptions:
%// Warnings:
%//*************************************************************************
*********
%// Order of elements
%//      -Define globals
%//      -Define constants
%//      -Define elements of input vector
%//      -Functions
%//*************************************************************************
*********
% Establishes missile dimensions for use in computing
% Aerodynamic forces and moments
% Except where noted, all dimensions in MKS system

% Missile Name:  STANDARD RIM-67 MR

% ******  define globals  ******
global m d L XCG XCPN XCPW XCPB XHL
global ST SW SPLAN SREF

% ******  define constants  ******
%------------ missile body dimensions -----------------------
m=621;                  % mass, may be time varying
d=.343;                  % diameter
L=4.554;                   % length
XCG=2.205;                 % initial c.g., may be time varying
LN=.728;                  % length of nose cone
%------------ tailplane dimensions --------------------------
XHL=4.351;                 % hinge line arm
CRT=.356;                 % tail root chord
CTT=.127;                 % tail tip chord
TXT=.0;                 % tail extension
HT=.383;                   % tail height
%-------------- wing dimensions -------------------------
XW=1.12;                  % wing to radome tangency point
CRW=2.314;                 % wing root chord
CTW=1.93;                    % wing tip chord
WXT=0;                   % wing extension
HW=.142;                 % wing height
```

```
% ******  define input vector  ******

% ******  initialize variables  ******

% ************  functions  ************
%-------------- centers of pressure -------------------------
XCPN=.67*LN;                                % nose CP
XCPW=LN+XW+.7*CRW-.2*CTW;                   % wing CP
AN=.67*LN*d;                                % plan area of nose
AB=(L-LN)*d;                                % plan area of body
XCPB=(.67*AN*LN+AB*(LN+.5*(L-LN)))/...      % body CP
   (AN+AB);
%-------------- area computation ----------------------------
SW=.5*HW*(CTW+CRW)+CRW*WXT;     % wing area
ST=.5*HT*(CTT+CRT)+CRT*TXT;     % tail area
SPLAN=(L-LN)*d+.67*L*d;         % body and nose plan area
SREF=pi*d^2/4;                  % missile cross section


%----------- compute the inertial matrix --------------------
r=d/2;
Jx=m*r^2/2;
Jy=m*(L^2/12+r^2/4);
Jz=Jy;

%//end of file missiledata3.m
```

```
%//*********************************************************************
*********
%// File:  missiledata4.m
%// Name:  LCDR Robert D. Broadston
%// MSEE/EE Thesis
%// Operating Environment:  Windows NT 4.0 Service Pack 5
%// Compiler:  MatLab v5.3
%// Date:  19 Sep 00
%// Description:  computes missile data for SM-2 ER
%// Inputs:  none
%// Outputs:  none
%// Process:
%// Assumptions:
%// Warnings:
%//*********************************************************************
*********
%// Order of elements
%//      -Define globals
%//      -Define constants
%//      -Define elements of input vector
%//      -Functions
%//*********************************************************************
*********
% Establishes missile dimensions for use in computing
% Aerodynamic forces and moments
% Except where noted, all dimensions in MKS system

% Missile Name:  STANDARD WITH BOOSTER

% ******  define globals  ******
global m d L XCG XCPN XCPW XCPB XHL
global ST SW SPLAN SREF

% ******  define constants  ******
%------------ missile body dimensions -----------------------
m=1680;                  % mass, may be time varying
d=.343;                  % diameter
L=7.976;                   % length
XCG=3.987;                 % initial c.g., may be time varying
LN=.728;                  % length of nose cone
%------------ tailplane dimensions --------------------------
XHL=7.6;                 % hinge line arm
CRT=.75;                 % tail root chord
CTT=.3;                 % tail tip chord
TXT=.0;                 % tail extension
HT=.65;                  % tail height
%--------------- wing dimensions --------------------------
XW=1.12;                  % wing to radome tangency point
CRW=2.7;                 % wing root chord
CTW=2.5;                    % wing tip chord
WXT=0;                   % wing extension
HW=.151;                 % wing height
```

```
% ******  define input vector  ******

% ******  initialize variables  ******

% ************  functions  ************
%-------------- centers of pressure -------------------------
XCPN=.67*LN;                              % nose CP
XCPW=LN+XW+.7*CRW-.2*CTW;                 % wing CP
AN=.67*LN*d;                              % plan area of nose
AB=(L-LN)*d;                              % plan area of body
XCPB=(.67*AN*LN+AB*(LN+.5*(L-LN)))/...    % body CP
   (AN+AB);
%-------------- area computation ----------------------------
SW=.5*HW*(CTW+CRW)+CRW*WXT;     % wing area
ST=.5*HT*(CTT+CRT)+CRT*TXT;     % tail area
SPLAN=(L-LN)*d+.67*L*d;         % body and nose plan area
SREF=pi*d^2/4;                  % missile cross section


%----------- compute the inertial matrix --------------------
r=d/2;
Jx=m*r^2/2;
Jy=m*(L^2/12+r^2/4);
Jz=Jy;

%//end of file missiledata4.m
```

```
function y=modelswitch(u)
%MODELSWITCH  Switches missile models to simulate
%             staging for TBM interceptor demo
%             MODELSWITCH(T)
%             see also
%        Copyright 1999-2000 by Triple B Enterprises

%//*********************************************************************
*********
%// File:  modelswitch.m
%// Name:  LCDR Robert D. Broadston
%// MSEE/EE Thesis
%// Operating Environment:  Windows NT 4.0 Service Pack 5
%// Compiler:  MatLab v5.3
%// Date:  19 Sep 2000
%// Description:  Switches between interceptor with booster
%//               and without booster for TBM demo
%// Inputs:  simulation time
%// Outputs:  none
%// Process:  none
%// Assumptions:
%// Warnings:
%//*********************************************************************
*********
%// Order of elements
%//      -Define globals
%//      -Define constants
%//      -Define elements of input vector
%//      -Functions
%//*********************************************************************
*********
% ******  define globals  ******

% ******  define constants  ******

% ******  define input vector  ******

% ******  initialize variables  ******

% ************  functions  ************
if (u>10.5)
   missiledata3;
else
   missiledata4;
end

%// end of file modelswitch.m
```

```
%//*****************************************************************
*********
%// File:  noisestudy.m
%// Name:  LCDR Robert D. Broadston
%// MSEE/EE Thesis
%// Operating Environment:  Windows NT 4.0 Service Pack 5
%// Compiler:  MatLab v5.3
%// Date:  19 Sep 00
%// Description:  performs noise study using modified AAM model
%//               thesisnoise.mdl
%// Inputs:
%// Outputs:
%// Process:
%// Assumptions:
%// Warnings:
%//*****************************************************************
*********
%// Order of elements
%//      -Define globals
%//      -Define constants
%//      -Define elements of input vector
%//      -Functions
%//*****************************************************************
*********
% ******  define globals  ******

% ******  define constants  ******

% ******  define input vector  ******

% ******  initialize variables  ******
% initialize simulation
thesisinit
% initialize variables
holdrange=[];
holdpos=[];
% initialize target
tgtinit=tgtset(42190,6000,45);
tic
```

```
% ************  functions  ************
% 100 realizations
for numloops=1:100
   disp(numloops)
   sim('thesisnoise')
   % analyze data from current run
   range=sqrt((MissileOut(:,1)-TgtOut(:,1)).^2+...
         (MissileOut(:,2)-TgtOut(:,3)).^2+...
         (MissileOut(:,3)-TgtOut(:,5)).^2);
   disp(min(range))
   holdrange(numloops)=min(range);
   idx=find(range==min(range));
   holdpos(numloops,:)=MissileOut(idx,1:3)-TgtOut(idx,1:2:5);
end
missdistance=mean(holdrange)
sigmadistance=std(holdrange)

figure(5)
plot3(holdpos(:,1),holdpos(:,2),holdpos(:,3),'*')

%//end of file noisestudy.m
```

```
function y=propnav3D(u)
%PROPNAV3D Computes exact proportional navigation
%          in three dimensions for full aero model
%        see also EXACTPROPNAV2
%        Copyright 1999-2000 by Triple B Enterprises
%//*****************************************************************
*********
%// File:  propnav3d.m
%// Name:  LCDR Robert D. Broadston
%// MSEE/EE Thesis
%// Operating Environment:  Windows NT 4.0 Service Pack 5
%// Compiler:  MatLab v5.3
%// Date:  12 Sep 00
%// Description:  3D prop nav law for full aero model
%// Inputs:  seeker output
%// Outputs:  command accelerations
%// Process:
%// Assumptions:
%// Warnings:
%//*****************************************************************
*********
%// Order of elements
%//      -Define globals
%//      -Define constants
%//      -Define elements of input vector
%//      -Functions
%//*****************************************************************
*********
% ******  define globals  ******
global satflag

% ******  define constants  ******
Nprime=5;
Nprimez=5;

% ******  define input vector  ******
thetadot=u(1);
phidot=u(2);
los=u(3);
philos=u(4);
R=u(6);
Vc=-u(5);
heading=u(7);
Vm=u(8);
Vmdot=u(9);
phi=u(10);
theta=u(11);
psi=u(12);
time=u(13);
```

```
% ******  initialize variables  ******

% ************  functions  ************
ny=Nprime*Vc*(thetadot)/cos(psi-los);
nz=Nprimez*Vc*(phidot)-9.8045;

% control force limiter
if satflag
   if (abs(ny)>30*9.8045)
      ny=sign(ny)*30*9.8045;
   end
   if (abs(nz)>30*9.8045)
      nz=sign(nz)*30*9.8045;
   end
end

y=[ny;nz];

%//end of file propnav3d.m
```

```
function y=propnavpt(u)
%PROPNAVPT Computes exact proportional navigation
%          with dragforce inputs for point mass simulation
%        see also EXACTPROPNAV2
%        Copyright 1999-2000 by Triple B Enterprises

%//*********************************************************************
*********
%// File:  PROPNAVPT.m
%// Name:  LCDR Robert D. Broadston
%// MSEE/EE Thesis
%// Operating Environment:  Windows NT 4.0 Service Pack 5
%// Compiler:  MatLab v5.3
%// Date:  24 May 2000
%// Description:  Proportional navigation guidance law for 6DOF
%//               flight model.  Computes applied forces for use
%//               by induced drag model.  Required to eliminate
%//               algebraic loops in the simulation
%// Inputs:  [seeker data,IMU data,timer]
%// Outputs:  [command accelerations,applied forces]
%// Process:  proportional navigation law
%// Assumptions:  none
%// Warnings:  none
%//*********************************************************************
*********
%// Order of elements
%//      -Define globals
%//      -Define constants
%//      -Define elements of input vector
%//      -Functions
%//*********************************************************************
*********
% ******  define globals  ******
global m satflag

% ******  define constants  ******
Nprime=5;
Nprimez=5;

% ******  define input vector  ******
thetadot=u(1);
phidot=u(2);
los=u(3);
philos=u(4);
R=u(6);
Vc=-u(5);
heading=u(7);
Vm=u(8);
Vmdot=u(9);
phi=u(10);
theta=u(11);
psi=u(12);
time=u(13);
```

```
% ******  initialize variables  ******

% ************  functions  ************
% classic PN guidance law
ny=Nprime*Vc*(thetadot)/cos(psi-los);
% vertical acceleration must account for gravity
nz=Nprimez*Vc*(phidot)/cos(theta-philos)-9.8045;

% control force limiter
if satflag
   if (abs(ny)>30*9.8045)
      ny=sign(ny)*30*9.8045;
   end
   if (abs(nz)>30*9.8045)
      nz=sign(nz)*30*9.8045;
   end
end

% compute ABC forces applied
Fx=0;
Fy=ny*m;
Fz=nz*m;

% output vector
y=[ny;nz;Fx;Fy;Fz];

%//end of file PROPNAVPT.m
```

```
function y=propnavtbm(u)
%PROPNAVTBM Computes exact proportional navigation
%          with dragforce inputs for point mass simulation
%        see also EXACTPROPNAV2
%        Copyright 1999-2000 by Triple B Enterprises

%//*********************************************************************
*********
%// File:  PROPNAVTBM.m
%// Name:  LCDR Robert D. Broadston
%// MSEE/EE Thesis
%// Operating Environment:  Windows NT 4.0 Service Pack 5
%// Compiler:  MatLab v5.3
%// Date:  24 May 2000
%// Description:  Proportional navigation guidance law for 6DOF
%//               flight model.  Computes applied forces for use
%//               by induced drag model.  Required to eliminate
%//               algebraic loops in the simulation
%// Inputs:  [seeker data,IMU data,timer]
%// Outputs:  [command accelerations,applied forces]
%// Process:  proportional navigation law
%// Assumptions:  none
%// Warnings:  none
%//*********************************************************************
*********
%// Order of elements
%//      -Define globals
%//      -Define constants
%//      -Define elements of input vector
%//      -Functions
%//*********************************************************************
*********
% ******  define globals  ******
global m satflag

% ******  define constants  ******
Nprime=5;
Nprimez=5;

% ******  define input vector  ******
thetadot=u(1);
phidot=u(2);
los=u(3);
philos=u(4);
R=u(6);
Vc=-u(5);
heading=u(7);
Vm=u(8);
Vmdot=u(9);
phi=u(10);
theta=u(11);
psi=u(12);
time=u(13);
```

```
% ******  initialize variables  ******

% ************  functions  ************
% classic PN guidance law
ny=Nprime*Vc*(thetadot);
% vertical acceleration
nz=Nprimez*Vc*(phidot);

% control force limiter
if satflag
   if (abs(ny)>30*9.8045)
      ny=sign(ny)*30*9.8045;
   end
   if (abs(nz)>30*9.8045)
      nz=sign(nz)*30*9.8045;
   end
end

% compute ABC forces applied
Fx=0;
Fy=ny*m;
Fz=nz*m;

% output vector
y=[ny;nz;Fx;Fy;Fz];

%//end of file propnavtbm.m
```

```
function y=q2euler(u)
%Q2EULER Computes Euler angles from quaternions
%        see also ALPHABETA
%        Copyright 1999-2000 by Triple B Enterprises
%//*********************************************************************
*********
%// File:  q2euler.m
%// Name:  LCDR Robert D. Broadston
%// MSEE/EE Thesis
%// Operating Environment:  Windows NT 4.0 Service Pack 5
%// Compiler:  MatLab v5.3
%// Date:  6 Apr 00
%// Description:  computes euler angles from quaternions
%// Inputs:  quaternion
%// Outputs:  euler angles
%// Process:  Kuiper
%// Assumptions:
%// Warnings:
%//*********************************************************************
*********
%// Order of elements
%//      -Define globals
%//      -Define constants
%//      -Define elements of input vector
%//      -Functions
%//*********************************************************************
*********
% ******  define globals  ******
% ******  define constants  ******
% ******  define input vector  ******
q0=u(1);
q1=u(2);
q2=u(3);
q3=u(4);

% ******  initialize variables  ******
% ************  functions  ************
% convert quaternions to Euler angles
m11=2*q0^2+2*q1^2-1;
m12=2*q1*q2+2*q0*q3;
m13=2*q1*q3-2*q0*q2;
m23=2*q2*q3+2*q0*q1;
m33=2*q0^2+2*q3^2-1;

psi=atan2(m12,m11);
theta=asin(-m13);
% correct for singularity in pitch
if (isreal(theta))
   theta=theta;
else
   theta=sign(-m13)*pi/2;
end
phi=atan2(m23,m33);
y=[phi,theta,psi];
%//end of file q2euler.m
```

```
function y=quat2b(y)
%QUAT2B  Computes rotation matrix from quaternions
%        QUAT2B(Y)
%        see also QUATERNION, B2QUAT
%        Copyright 1999-2000 by Triple B Enterprises
%//*********************************************************************
*********
%// File:  quat2b.m
%// Name:  LCDR Robert D. Broadston
%// MSEE/EE Thesis
%// Operating Environment:  Windows NT 4.0 Service Pack 5
%// Compiler:  MatLab v5.3
%// Date:  8 Dec 99
%// Description:  computes rotation matrix from quaternions
%// Inputs:  quaternion
%// Outputs:  rotation matrix B
%// Process:  Kuiper
%// Assumptions:
%// Warnings:
%//*********************************************************************
*********
%// Order of elements
%//      -Define globals
%//      -Define constants
%//      -Define elements of input vector
%//      -Functions
%//*********************************************************************
*********
% ******  define globals  ******

% ******  define constants  ******

% ******  define input vector  ******
q0=y(1);
q1=y(2);
q2=y(3);
q3=y(4);

% ******  initialize variables  ******

% ************  functions  ************
y=[q0^2+q1^2-q2^2-q3^2,  2*(q1*q2+q0*q3),      2*(q1*q3-q0*q2);
   2*(q1*q2-q0*q3),    q0^2-q1^2+q2^2-q3^2,    2*(q2*q3+q0*q1);
   2*(q1*q3+q0*q2),     2*(q2*q3-q0*q1),   q0^2-q1^2-q2^2+q3^2];

%//end of file quat2b.m
```

```
function y=quaternion(phi,theta,psi)
%QUATERNION    Computes quaternions from Euler angles
%              QUATERNION(PHI,THETA,PSI)
%        see also B2QUAT
%        Copyright 1999-2000 by Triple B Enterprises
%//*****************************************************************
*********
%// File:  quaternion.m
%// Name:  LCDR Robert D. Broadston
%// MSEE/EE Thesis
%// Operating Environment:  Windows NT 4.0 Service Pack 5
%// Compiler:  MatLab v5.3
%// Date:  8 Aug 00
%// Description:  computes quaternion from euler angles
%// Inputs:  euler angles
%// Outputs:  quaternion
%// Process:  Kuiper
%// Assumptions:
%// Warnings:
%//*****************************************************************
*********
%// Order of elements
%//      -Define globals
%//      -Define constants
%//      -Define elements of input vector
%//      -Functions
%//*****************************************************************
*********
% ******  define globals  ******

% ******  define constants  ******

% ******  define input vector  ******

% ******  initialize variables  ******
q0=cos(phi/2)*cos(theta/2)*cos(psi/2)+...
   sin(phi/2)*sin(theta/2)*sin(psi/2);

q1=sin(phi/2)*cos(theta/2)*cos(psi/2)-...
   cos(phi/2)*sin(theta/2)*sin(psi/2);

q2=cos(phi/2)*sin(theta/2)*cos(psi/2)+...
   sin(phi/2)*cos(theta/2)*sin(psi/2);

q3=cos(phi/2)*cos(theta/2)*sin(psi/2)-...
   sin(phi/2)*sin(theta/2)*cos(psi/2);


% ************  functions  ************
y=[q0;q1;q2;q3];

%//end of file quaternion.m
```

```
function y=rhovalt(alt)
%RHOVALT Computes atmospheric density vs altitude
%        for ICAO standard atmosphere
%        RHOVALT(ALT)
%        see also MACHVALT,CDVMACH
%        Copyright 1999-2000 by Triple B Enterprises
%//*****************************************************************
*********
%// File:  rhovalt.m
%// Name:  LCDR Robert D. Broadston
%// MSEE/EE Thesis
%// Operating Environment:  Windows NT 4.0 Service Pack 5
%// Compiler:  MatLab v5.3
%// Date:  8 Jun 00
%// Description:  computes atmospheric density from ICAO standard
%//               atmosphere.  exponential model
%// Inputs:   altitude
%// Outputs:  rho
%// Process:
%// Assumptions:
%// Warnings:
%//*****************************************************************
*********
%// Order of elements
%//      -Define globals
%//      -Define constants
%//      -Define elements of input vector
%//      -Functions
%//*****************************************************************
*********
% ******  define globals  ******

% ******  define constants  ******

% ******  define input vector  ******

% ******  initialize variables  ******
alt=abs(alt);                    % account for NED coord

% ************  functions  ************
if alt>9144
   y=1.75228763*exp(-alt/6705.6);
else
   y=1.22557*exp(-alt/9144);
end

%//end of file rhovalt.m
```

```
function y=sixdofdyn(u)
%SIXDOFDYN Computes motion dymanics for six degrees
%          of freedom
%        see also FLATEARTHDYN
%        Copyright 1999-2000 by Triple B Enterprises
%//*****************************************************************
*********
%// File:  sixdofdyn.m
%// Name:  LCDR Robert D. Broadston
%// MSEE/EE Thesis
%// Operating Environment:  Windows NT 4.0 Service Pack 5
%// Compiler:  MatLab v5.3
%// Date:  7 Apr 00
%// Description:  computes 6DOF dynamics in ECI coordinates
%// Inputs:  state vector
%// Outputs:  derivative of state vector
%// Process:  Stevens & Lewis
%// Assumptions:
%// Warnings:
%//*****************************************************************
*********
%// Order of elements
%//      -Define globals
%//      -Define constants
%//      -Define elements of input vector
%//      -Functions
%//*****************************************************************
*********
% ******  define globals  ******

% ******  define constants  ******
omega_x=7.292115e-5;      % earth rotation rate
GM_E=3.9860014e14;        % G*mass of earth
r_E=6.378164e6;           % radius of earth
f=1/298.257;              % ellipsoidal squash factor

% ******  define input vector  ******
p=[u(1);u(2);u(3)];
v_b=[u(4);u(5);u(6)];
omega_b=[u(7);u(8);u(9)];
P=u(7);  Q=u(8);  R=u(9);

q=[u(10);u(11);u(12);u(13)];
magq=norm(q,2);
q=q/magq;

x=[p;v_b;omega_b;q];

% inertial matrix
J=[u(14),0,0;
   0,u(15),0;
   0,0,u(16)];

% forces
F_B=[u(17);u(18);u(19)];
```

```
% torques
T_B=[u(20);u(21);u(22)];

% the ever lovin' force of gravity
g_p=[u(23);u(24);u(25)];

% and lest we forget, mass
m=u(26);

% ******  initialize variables  ******
omega_E=[omega_x;0;0];       % earth rotational velocity vector
OMEGA_E=[0,0,0;0,0,-omega_x;0,omega_x,0];

% compute rotation matrices
B=quat2b(q);

OMEGA_B=[0 -R  Q;
         R  0 -P;
        -Q  P  0];

OMEGA_q=[0  P  Q  R;
        -P  0 -R  Q;
        -Q  R  0 -P;
        -R -Q  P  0];

% ************  functions  ************
y=[OMEGA_E,         B',                    zeros(3),      zeros(3,4);
   -B*OMEGA_E^2, -(OMEGA_B+B*OMEGA_E*B'),  zeros(3),      zeros(3,4);
   zeros(3),      zeros(3),           -1*inv(J)*OMEGA_B*J, zeros(3,4);
   zeros(4,3),    zeros(4,3),             zeros(4,3),  -(1/2)*OMEGA_q];
y=y*x;
y=y+[zeros(3,1);
   B*g_p+(1/m)*F_B;
   inv(J)*T_B;
   zeros(4,1)];

%//end of file sixdofdyn.m
```

```
%//*************************************************************************
*********
%// File:  spielberg.m
%// Name:  LCDR Robert D. Broadston
%// MSEE/EE Thesis
%// Operating Environment:  Windows NT 4.0 Service Pack 5
%// Compiler:  MatLab v5.3
%// Date:  20 Sep 00
%// Description:  creates a movie of AAM simulation using
%//               thesism
%// Inputs:  none
%// Outputs:  none
%// Process:
%// Assumptions:
%// Warnings:
%//*************************************************************************
*********
%// Order of elements
%//      -Define globals
%//      -Define constants
%//      -Define elements of input vector
%//      -Functions
%//*************************************************************************
*********
% ******  define globals  ******

% ******  define constants  ******

% ******  define input vector  ******

% ******  initialize variables  ******
thesisinit

% ************  functions  ************
figure(1)
clf
tgtinit=tgtset(40000,5000,135);
target_turn=12;

for timestep=1:44
   tmax=.25*(timestep+0)
   sim('thesism')
   l=length(MissileOut);
   % a little 3D action for the fans
   figure(1)
   clf
   view(-30,25)
   hold on
   axis([0 40000 0 8000 4500 6500])
```

```
% plot commands
   plot3(MissileOut(:,1),MissileOut(:,2),-MissileOut(:,3),...
      TgtOut(:,1),TgtOut(:,3),-TgtOut(:,5))

   plot3(MissileOut(l,1),MissileOut(l,2),-MissileOut(l,3),'*',...
      TgtOut(l,1),TgtOut(l,3),-TgtOut(l,5),'x')

   velx=[0 0]; vely=[7000 7000]; velz=[4500 1.5*MissileV(l)+4500];
   plot3(velx,vely,velz,'r.-')
   ncyx=[0 0]; ncyy=[6500 6500]; ncyz=[4500 7*abs(omegaout(l,1))+4500];
   plot3(ncyx,ncyy,ncyz,'g.-')
   nczx=[0 0]; nczy=[6000 6000]; nczz=[4500 7*abs(omegaout(l,2))+4500];
   plot3(nczx,nczy,nczz,'b.-')
   if (tmax==round(tmax))
      plot3([MissileOut(l,1) TgtOut(l,1)],[MissileOut(l,2)
TgtOut(l,3)],...
         [-MissileOut(l,3) -TgtOut(l,5)],'k')
   end

   hold off
   grid on

   title(['Missile Engagement ',date],'FontSize',18)
   text(35000,6000,6000,[num2str(tmax,'%2.2f') '
seconds'],'FontSize',18)
   text(0,7000,1.5*MissileV(l)+4500,[' '
num2str(MissileV(l),'%4.0f')],'FontSize',14)
   text(0,6500,7*abs(omegaout(l,1))+4500,[' '
num2str(omegaout(l,1)/9.8045,'%2.1f')],'FontSize',14)
   text(0,6000,7*abs(omegaout(l,2))+4500,['   '
num2str(omegaout(l,2)/9.8045,'%2.1f')],'FontSize',14)

   M(timestep)=getframe;
end
movie(M)

%//end of file spielberg.m
```

```
function y=tgo(u)
%TGO    Computes time to go from Range and Range Rate
%        see also
%        Copyright 1999-2000 by Triple B Enterprises
%//*****************************************************************
*********
%// File:  tgo.m
%// Name:  LCDR Robert D. Broadston
%// MSEE/EE Thesis
%// Operating Environment:  Windows NT 4.0 Service Pack 5
%// Compiler:  MatLab v5.3
%// Date:  31 May 00
%// Description:  computes tgo
%// Inputs:  range, range rate
%// Outputs:  tgo
%// Process:
%// Assumptions:
%// Warnings:
%//*****************************************************************
*********
%// Order of elements
%//      -Define globals
%//      -Define constants
%//      -Define elements of input vector
%//      -Functions
%//*****************************************************************
*********
% ******  define globals  ******

% ******  define constants  ******

% ******  define input vector  ******

% ******  initialize variables  ******

% ************  functions  ************
if (u(2)==0)
   y=100;
else
   y=abs(u(1)/u(2));
end

%//end of file tgo.m
```

```
function y=tgtset(Range,Alt,Hdg)
%TGTSET  Sets tgtinit variable for missile simulations
%        default tgt speed set at .83 Mach.  Enter altitude
%        as a positive number
%        TGTSET(RANGE,ALT,HDG)
%        see also QUATERNION, BQUAT
%        Copyright 1999-2000 by Triple B Enterprises
%//*****************************************************************
*********
%// File:  tgtset.m
%// Name:  LCDR Robert D. Broadston
%// MSEE/EE Thesis
%// Operating Environment:  Windows NT 4.0 Service Pack 5
%// Compiler:  MatLab v5.3
%// Date:  8 Jun 00
%// Description:  initializes target state vector
%// Inputs:  see above
%// Outputs:  target state vector
%// Process:
%// Assumptions:
%// Warnings:
%//*****************************************************************
*********
%// Order of elements
%//      -Define globals
%//      -Define constants
%//      -Define elements of input vector
%//      -Functions
%//*****************************************************************
*********
% ******  define globals  ******

% ******  define constants  ******
tgtmach=.83;                    % user sets Mach #

% ******  define input vector  ******

% ******  initialize variables  ******
tgtspd=tgtmach*machvalt(Alt);% machine computes speed

% ************  functions  ************
y=[Range;cos(Hdg*pi/180)*tgtspd;0;sin(Hdg*pi/180)*tgtspd;-Alt;0];
% Note:  negative altitude is for NED coords

%//end of file tgtset.m
```

```
function y=thebigstop(u)
%THEBIGSTOP consolidated simulation stop function
%
%        see also
%        Copyright 1999-2000 by Triple B Enterprises
%//*****************************************************************
*********
%// File:  thebigstop.m
%// Name:  LCDR Robert D. Broadston
%// MSEE/EE Thesis
%// Operating Environment:  Windows NT 4.0 Service Pack 5
%// Compiler:  MatLab v5.3
%// Date:  17 Sep 00
%// Description:  stops simulation under variety of conditions
%// Inputs:  see below
%// Outputs:  stop flag
%// Process:
%// Assumptions:
%// Warnings:
%//*****************************************************************
*********
%// Order of elements
%//      -Define globals
%//      -Define constants
%//      -Define elements of input vector
%//      -Functions
%//*****************************************************************
*********
% ******  define globals  ******
global stopflag

% ******  define constants  ******

% ******  define input vector  ******
R=u(1);
Rdot=u(2);
Vm=u(3);
Vt=u(4);
G=u(5);
Ny=u(6);
Nz=u(7);
time=u(8);

% ******  initialize variables  ******
stop=[];
y=0;
```

```
% ************  functions  ************
% check cases
%if (G>700)
%   y=111;
%   stop='G stop';
%end

if ((time>2.0)&(Vm<Vt))
   y=111;
   stop='V stop';
end

if ((time>2.0)&(Rdot>0))
   y=111;
   stop='Rdot stop';
end

if (R<1e-6)
   y=111;
   stop='R stop';
end

if ((Nz>30*9.8045)|(Ny>30*9.8045))
   y=111;
   stop='Cmd stop';
end
if stopflag
   disp(stop)
end

%//end of file thebigstop.m
```

```
%//*************************************************************************
*********
%// File:  thesis2plot.m
%// Name:  LCDR Robert D. Broadston
%// MSEE/EE Thesis
%// Operating Environment:  Windows NT 4.0 Service Pack 5
%// Compiler:  MatLab v5.3
%// Date:
%// Description:  Plots a variety of data for missile tracking
%//                simulations for use in thesis paper
%// Inputs:  none
%// Outputs:  purdy pitchers
%// Process:
%// Assumptions:
%// Warnings:
%//*************************************************************************
*********
%// Order of elements
%//      -Define globals
%//      -Define constants
%//      -Define elements of input vector
%//      -Functions
%//*************************************************************************
*********
% ******  define globals  ******

% ******  define constants  ******

% ******  define input vector  ******

% ******  initialize variables  ******
time=rem(now,1);
hr=floor(time*24);
mins=floor(rem(time*24,1)*60);
timestr=[' ',num2str(hr),':',num2str(mins)];

% ************  functions  ************
% engagement geometry
figure(1)
subplot(2,1,1)
plot(TgtOut(:,1),TgtOut(:,3),':',MissileOut(:,1),MissileOut(:,2))
axis equal
outtext1=['time: ',num2str(ip),' seconds'];
outtext2=['range: ',num2str(min(range)),' meters'];
text(300,4000,'Intercept at:')
text(300,3500,outtext1)
text(300,3000,outtext2)

title('Engagement Geometry')
xlabel('x (meters)')
ylabel('y (meters)')
legend('Target','Missile')
```

```
% missile to target distance
range=sqrt((MissileOut(:,1)-TgtOut(:,1)).^2+...
   (MissileOut(:,2)-TgtOut(:,3)).^2+(MissileOut(:,3)-TgtOut(:,5)).^2);
t=MissileOut(:,14);
t_disc=0:FILTSAMP:max(t);
index=find(range==min(range));
ip=t(index(1));

subplot(2,1,2)
plot(t,MissileV)
title('Missile to Target Range')
xlabel(['time (seconds)          ',date,timestr])
ylabel('missile velocity (m/s)')

% missile accelerations
gforce=sqrt(AccelOut(:,1).^2+AccelOut(:,2).^2 ...
   +AccelOut(:,3).^2)./9.8045;
figure(2)
subplot(2,1,1)
plot(t,gforce)
title('Missile Accelerations')

ylabel('Acceleration (g)')
axis([0 round(max(t)) 0 50])
% compute cost function J=20*e(tf)^2+integ(u^2)/200
u2=(omegaout(:,1).^2+omegaout(:,2).^2);
integral=0;
for ii=2:index
   integral=integral+(t(ii)-t(ii-1))*u2(ii-1);
end
J=20*min(range)^2+integral/1000;
%endfor
outtxt=['Missile divert:  ',num2str(integral)];
xlabel(outtxt)
% guidance command
subplot(2,1,2)
plot(t,omegaout(:,1),t,omegaout(:,2),':')
title('Guidance law command output')
outtxt=['Cost J: ',num2str(J),'   time (seconds)'];
xlabel([outtxt,'      ',date,timestr])
ylabel('n_c (m/sec^2)')
axis([0 round(max(t)) -300 300])
legend('n_c y','n_c z')

%//end of file thesis2plot.m
```

```
%//*************************************************************************
*********
%// File:  thesisinit.m
%// Name:  LCDR Robert D. Broadston
%// MSEE/EE Thesis
%// Operating Environment:  Windows NT 4.0 Service Pack 5
%// Compiler:  MatLab v5.3
%// Date:  5 Sep 00
%// Description:  This script file initializes thesis work
%//               missile simulation
%// Inputs:  none
%// Outputs:  none
%// Process:
%// Assumptions:
%// Warnings:
%//*************************************************************************
*********
%// Order of elements
%//      -Define globals
%//      -Define constants
%//      -Define elements of input vector
%//      -Functions
%//*************************************************************************
*********
clear
% ******  define globals  ******
global stopflag satflag XLAST FILTSAMP

% ******  define constants  ******
% physical constants
omega_x=7.292115e-5;     % earth rotation rate
GM_E=3.9860014e14;       % G*mass of earth
r_E=6.378164e6;          % radius of earth
f=1/298.257;             % ellipsoidal squash factor
omega_E=[omega_x;0;0];   % earth rotational velocity vector

% ******  define input vector  ******

% ******  initialize variables  ******
% missile physical parameters
MissileData;
% drawmissile;

% missile velocity vector
vB=[270;0;0];

% initial missile position vector
p=[0,0,-6000]';        % note altitude is negative in NED coord
t=0;

% compute Euler angles
psi=0*pi/180;
theta=0*pi/180;
phi=0*pi/180;
```

```
% ************  functions  ************
q_0=quaternion(phi,theta,psi);
q_0=q_0/sqrt(q_0(1)^2+q_0(2)^2+q_0(3)^2+q_0(4)^2);

B=quat2b(q_0);

P=0*pi/180;
Q=0*pi/180;
R=0*pi/180;

omega_B=[P;Q;R];

% initial state vector
init=[p;vB;omega_B;q_0];

% target initial state vector
tgtinit=[25000;
   -250;
   0;
   250;
   -6000;
   0];

tmax=200;
target_turn=0;        % set target turn rate, default=0
satflag=1;            % enable saturation of cmd accel
XLAST=[25000;         % initialize filter
   -250;
   0;
   0;
   250;
   0;
   -6000;
   0;
   0];
FILTSAMP=.1;         % set filter sample interval

%//end of file thesisinit.m
```

```
function y=vcpropnavpt(u)
%PROPNAVPT Computes exact proportional navigation
%          with dragforce inputs for point mass simulation
%        see also EXACTPROPNAV2
%        Copyright 1999-2000 by Triple B Enterprises

%//*********************************************************************
*********
%// File:  VCPROPNAVPT.m
%// Name:  LCDR Robert D. Broadston
%// MSEE/EE Thesis
%// Operating Environment:  Windows NT 4.0 Service Pack 5
%// Compiler:  MatLab v5.3
%// Date:  24 May 2000
%// Description:  Proportional navigation guidance law for 6DOF
%//               flight model.  Computes applied forces for use
%//               by induced drag model.  Required to eliminate
%//               algebraic loops in the simulation
%// Inputs:  [seeker data,IMU data,timer]
%// Outputs:  [command accelerations,applied forces]
%// Process:  proportional navigation law
%// Assumptions:  none
%// Warnings:  none
%//*********************************************************************
*********
%// Order of elements
%//      -Define globals
%//      -Define constants
%//      -Define elements of input vector
%//      -Functions
%//*********************************************************************
*********
% ******  define globals  ******
global m satflag

% ******  define constants  ******
Nprime=5;
Nprimez=5;

% ******  define input vector  ******
thetadot=u(1);
phidot=u(2);
los=u(3);
philos=u(4);
R=u(6);
Vc=-u(5);
heading=u(7);
Vm=u(8);
Vmdot=u(9);
phi=u(10);
theta=u(11);
psi=u(12);
time=u(13);
```

```
% ******  initialize variables  ******

% ************  functions  ************
ny=Nprime*750*(thetadot)/cos(psi-los)-Vmdot*tan(psi-los);
nz=Nprimez*750*(phidot)/cos(theta-philos)-Vmdot*tan(theta-los)-9.8045;

% control force limiter
if satflag
   if (abs(ny)>30*9.8045)
      ny=sign(ny)*30*9.8045;
   end
   if (abs(nz)>30*9.8045)
      nz=sign(nz)*30*9.8045;
   end
end

% compute ABC forces applied
Fx=0;
Fy=ny*m;
Fz=nz*m;

% output vector
y=[ny;nz;Fx;Fy;Fz];

%//end of file PROPNAVPT.m
```

APPENDIX C. SIMULATION DATA

This appendix contains the plots listed below for each guidance law. The engagement geometry is the same for each guidance law, initial range; 20 km, attack azimuth 45 degrees, co-altitude; 6,000 meters, 6 g target maneuver at 3 seconds t_{go}.

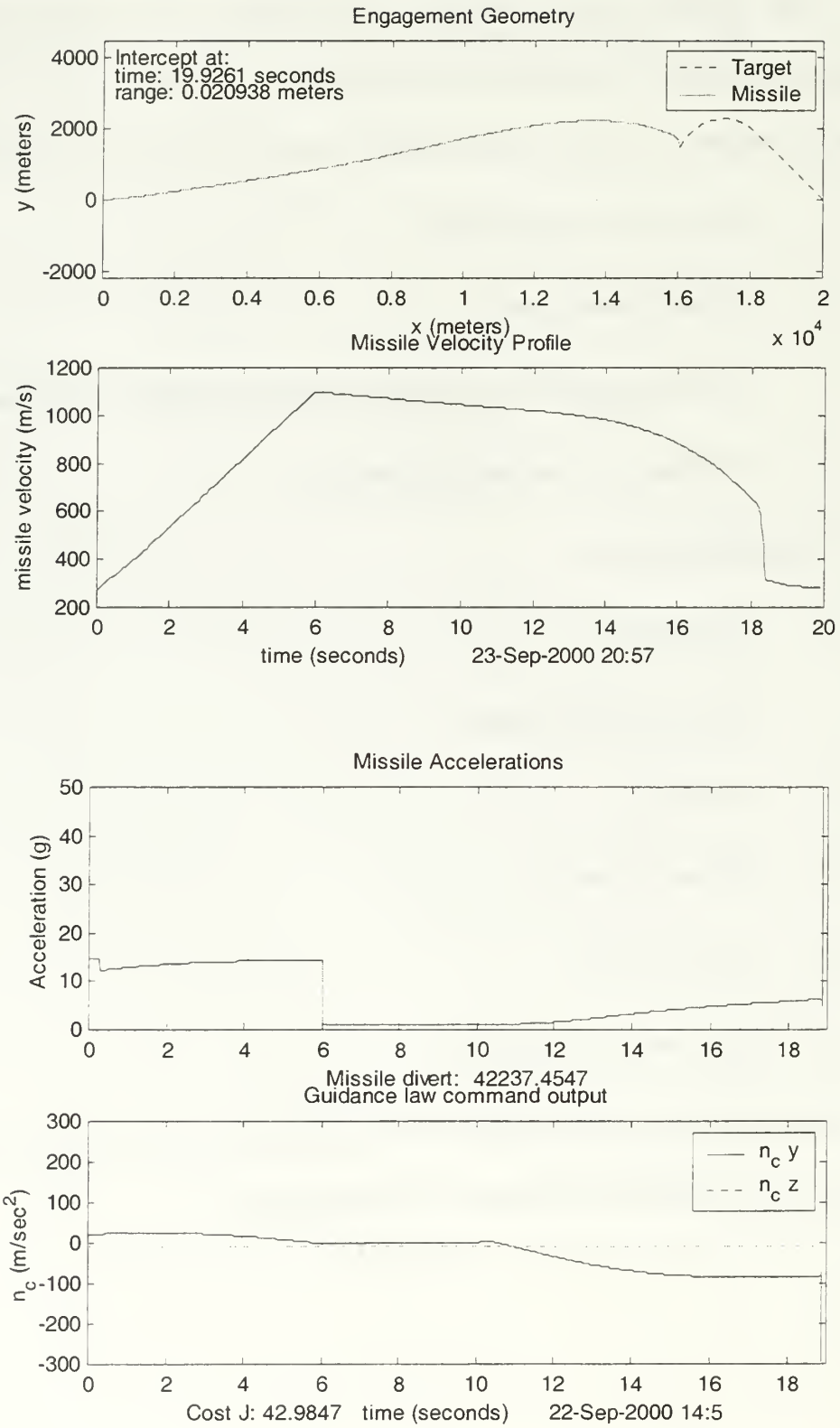
1. Plan view of engagement
2. Missile velocity profile
3. Missile accelerations and target acceleration estimates for filtered laws
4. Guidance law command accelerations

The guidance laws are:

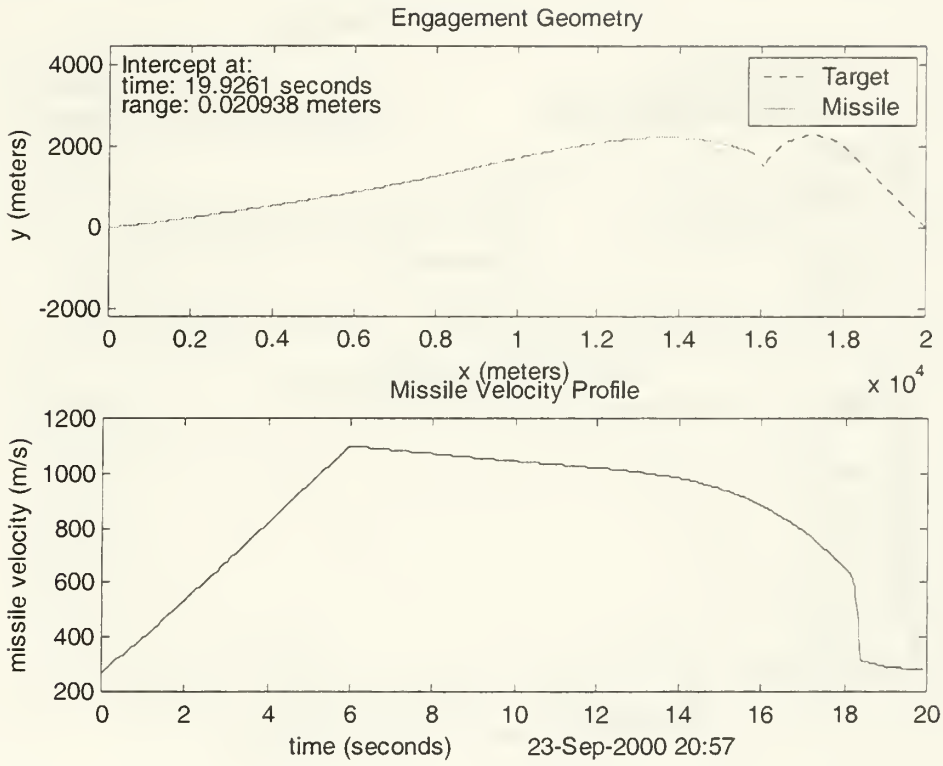
1. PN with N'=5
2. VCPN with constant gain
3. Bang-bang
4. Differential games
5. APN with Λ=5
6. Noisy seeker, PN with N'=5

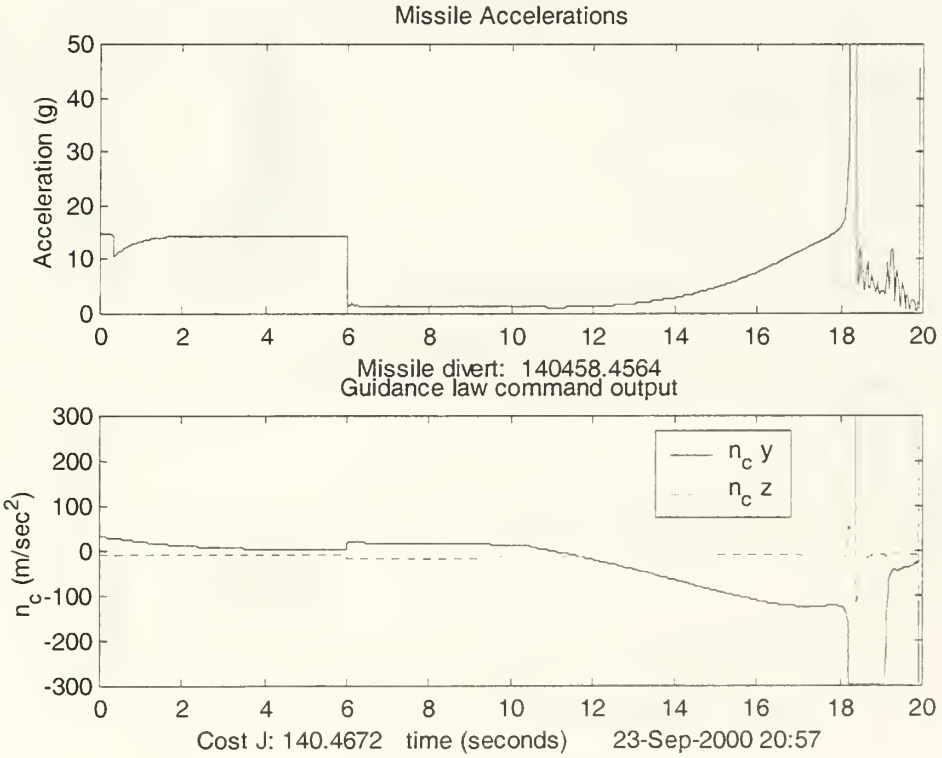
This appendix also contains plots from the full aerodynamic model running in an open loop with control surface deflections as the control input

A. PN (N'=5)

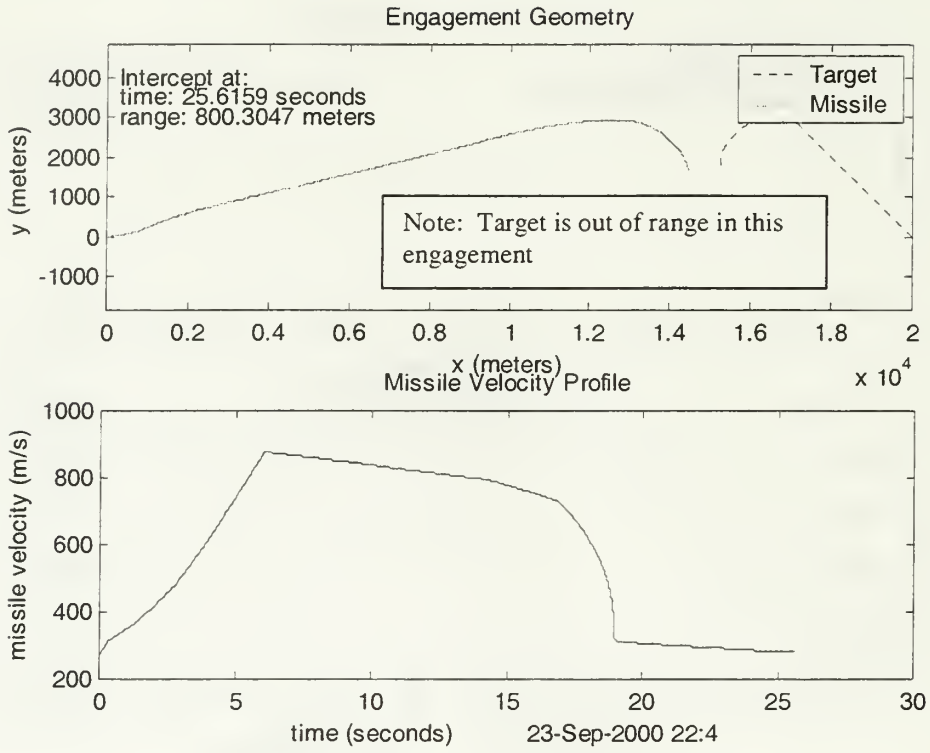

Engagement Geometry
Intercept at:
time: 19.9261 seconds
range: 0.020938 meters
Target
Missile
y (meters)
x (meters)
x 10^4
Missile Velocity Profile
missile velocity (m/s)
time (seconds)
23-Sep-2000 20:57
Missile Accelerations
Acceleration (g)
Missile divert: 42237.4547
Guidance law command output
n_c y
n_c z
n_c (m/sec^2)
Cost J: 42.9847
time (seconds)
22-Sep-2000 14:5

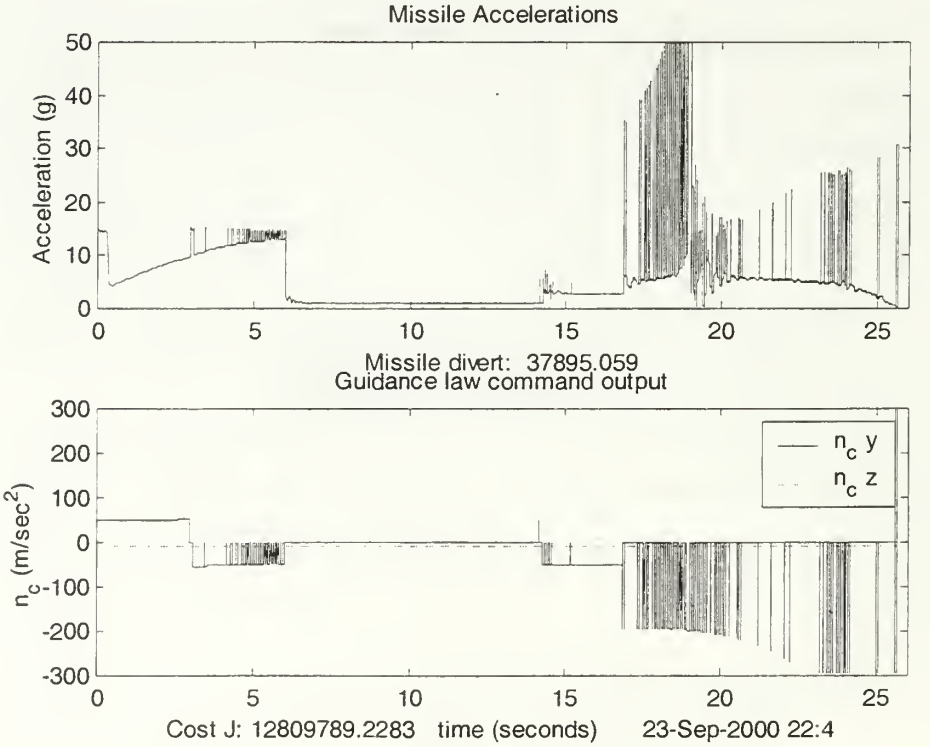
B. VCPN WITH CONSTANT GAIN



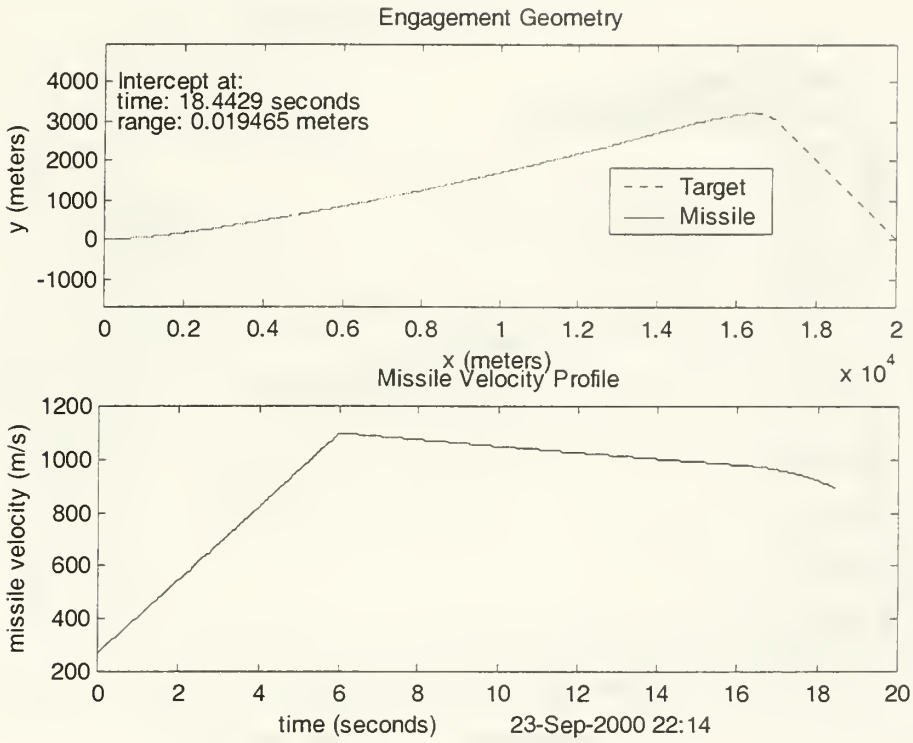


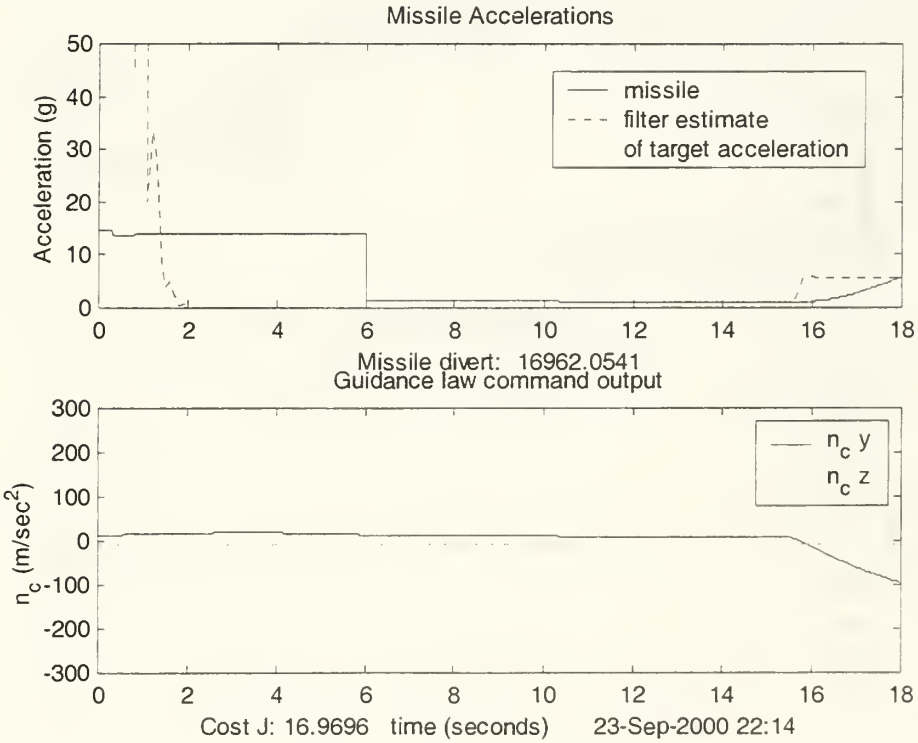
C. BANG-BANG


Engagement Geometry
Intercept at:
time: 25.6159 seconds
range: 800.3047 meters
Target
Missile
Note: Target is out of range in this engagement
y (meters)
x (meters)
x 10^4
Missile Velocity Profile
missile velocity (m/s)
time (seconds)
23-Sep-2000 22:4

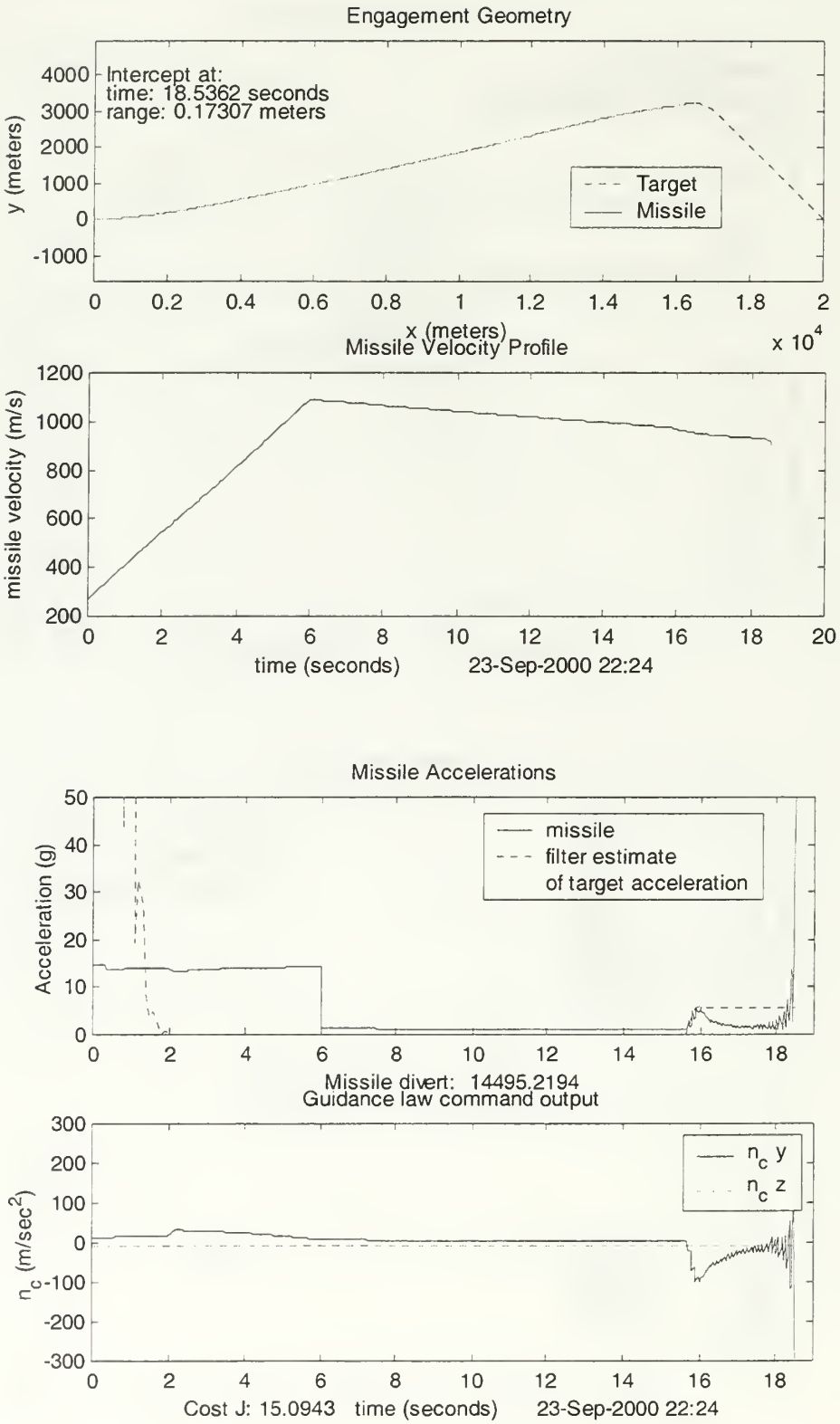

Missile Accelerations
Acceleration (g)
Missile divert: 37895.059
Guidance law command output
n_c y
n_c z
n_c (m/sec^2)
Cost J: 12809789.2283
time (seconds)
23-Sep-2000 22:4

D. DIFFERENTIAL GAMES

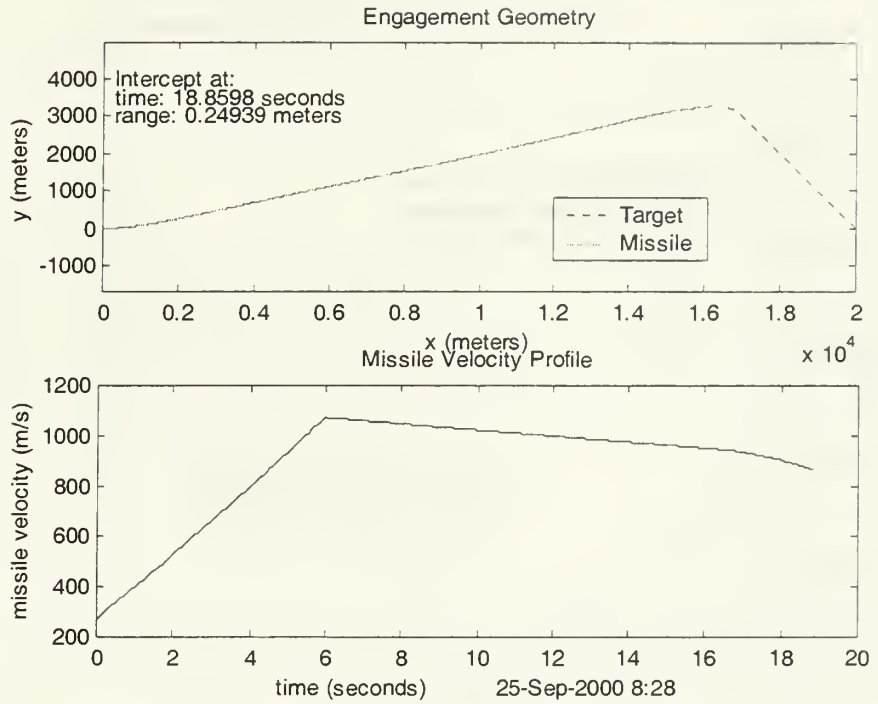


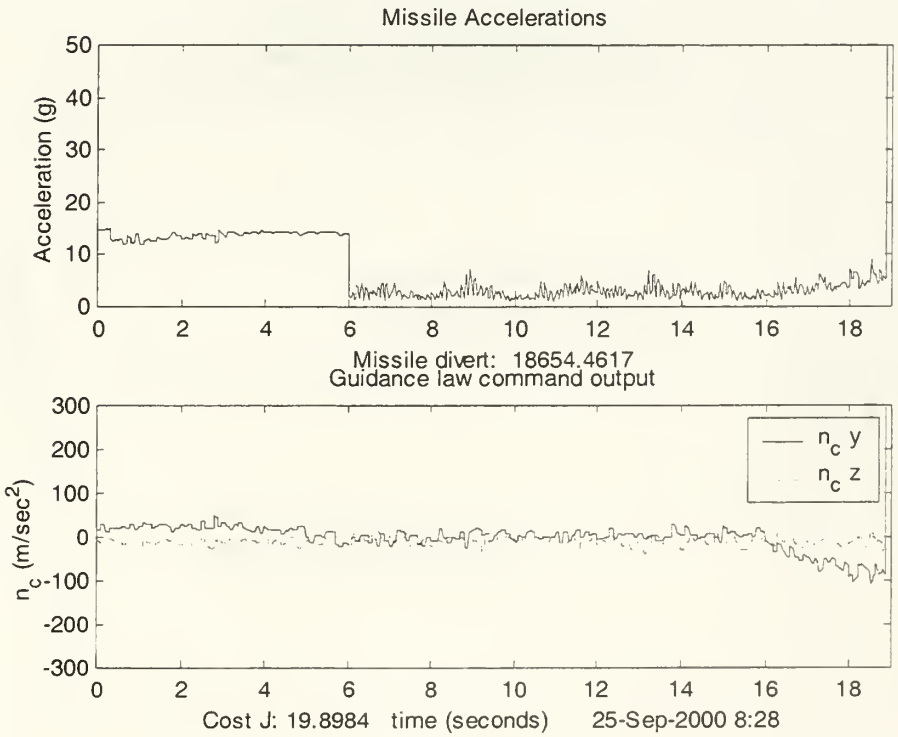


E. APN WITH Λ=5

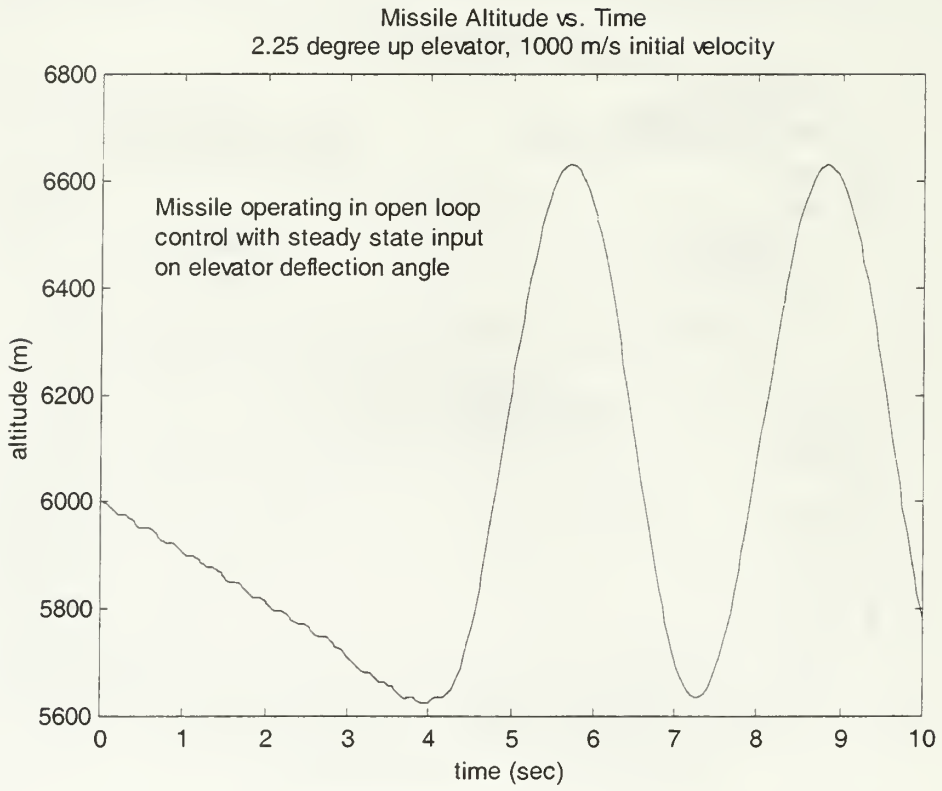


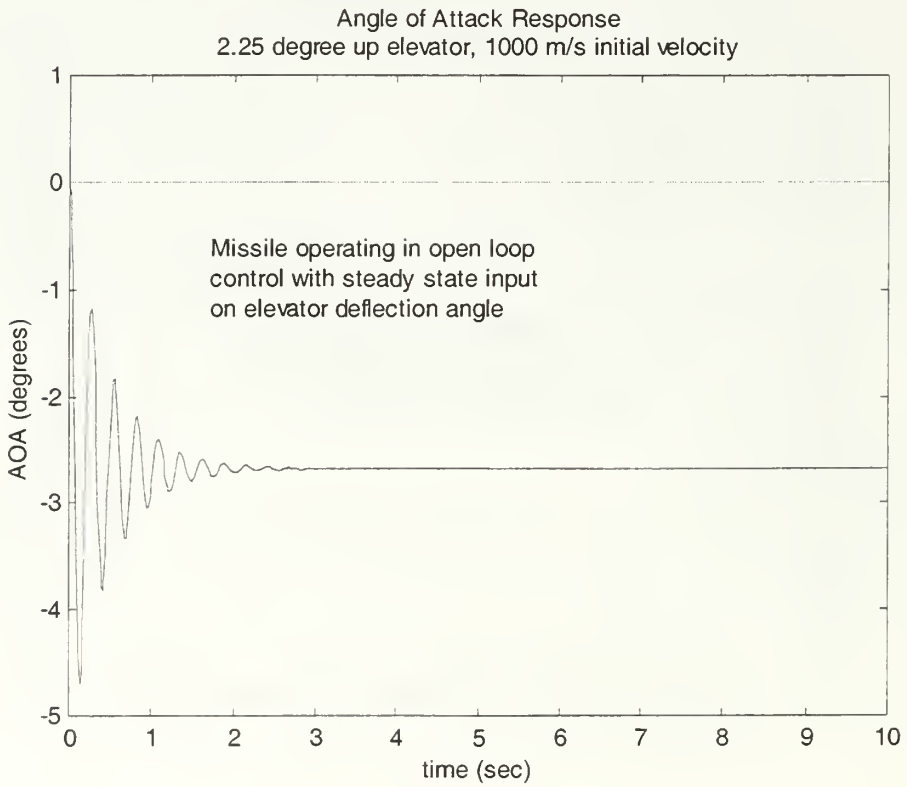
F. NOISY SEEKER, PN (*N*'=5)

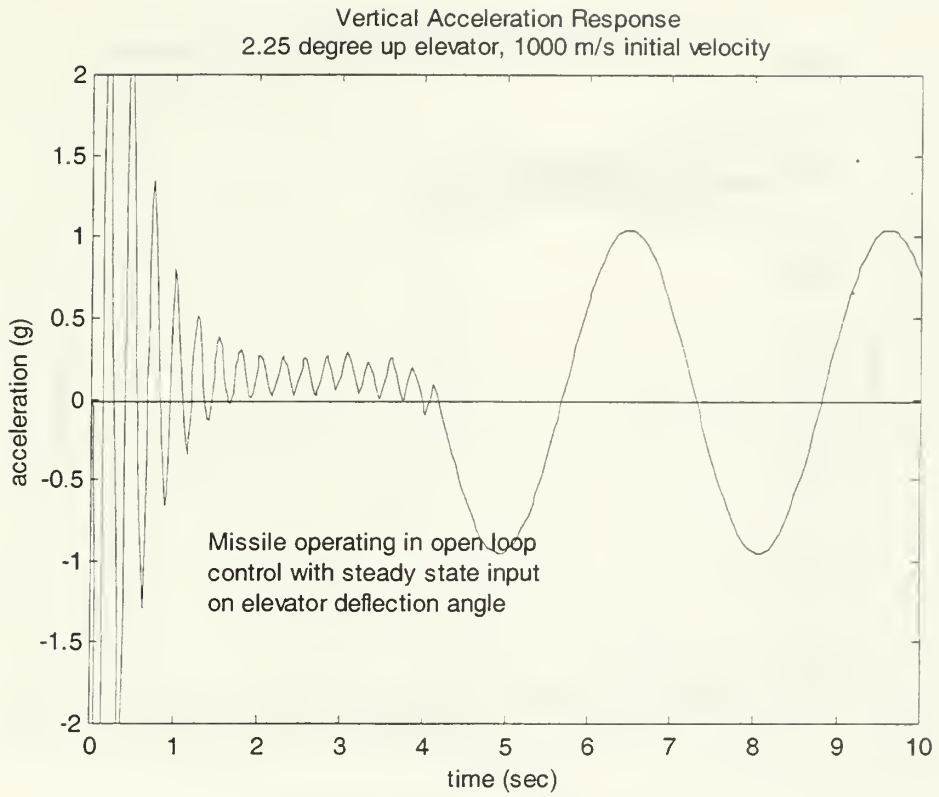


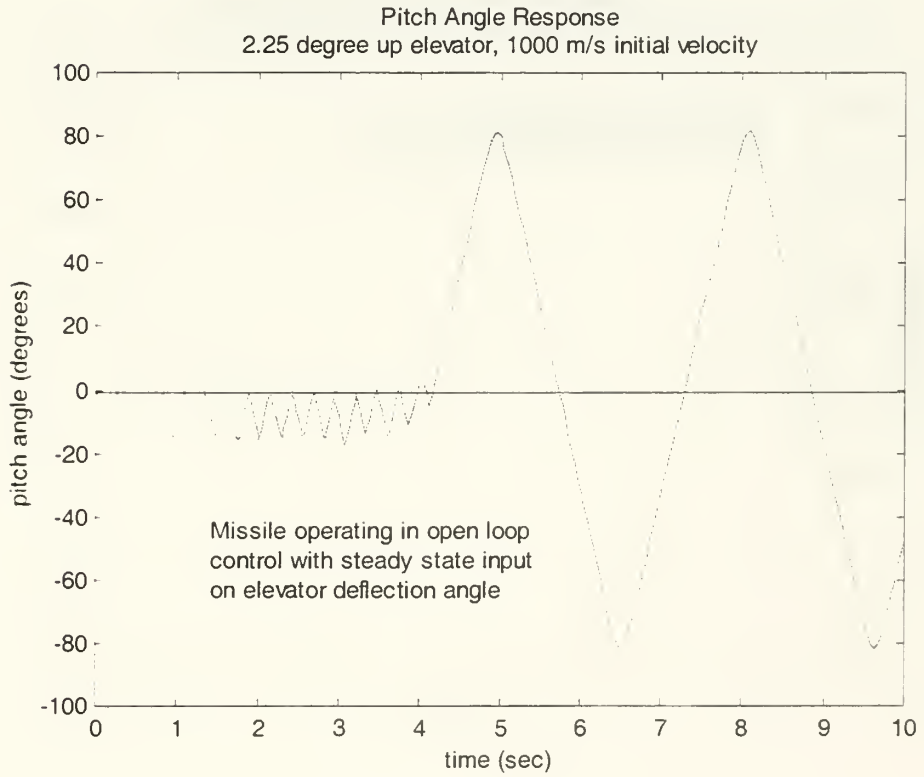


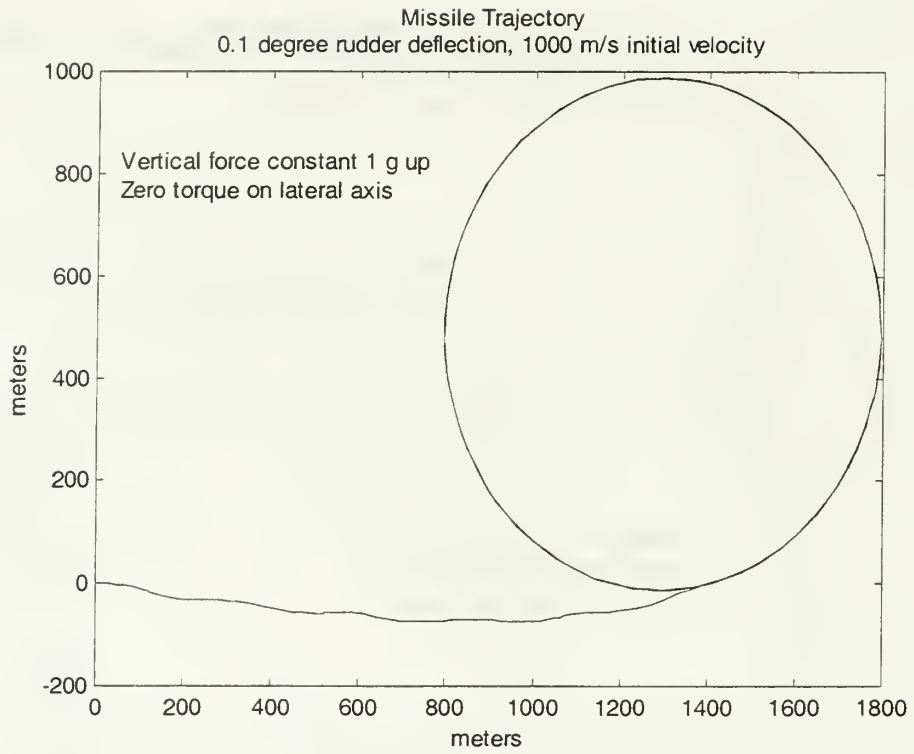
G. FULL AERODYNAMIC MODEL

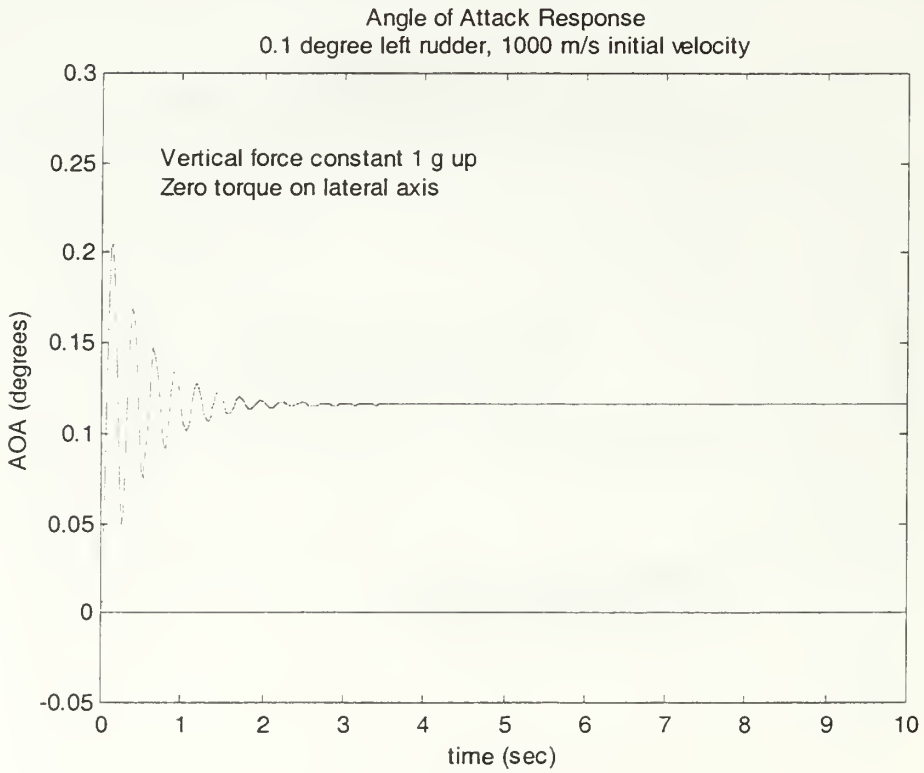


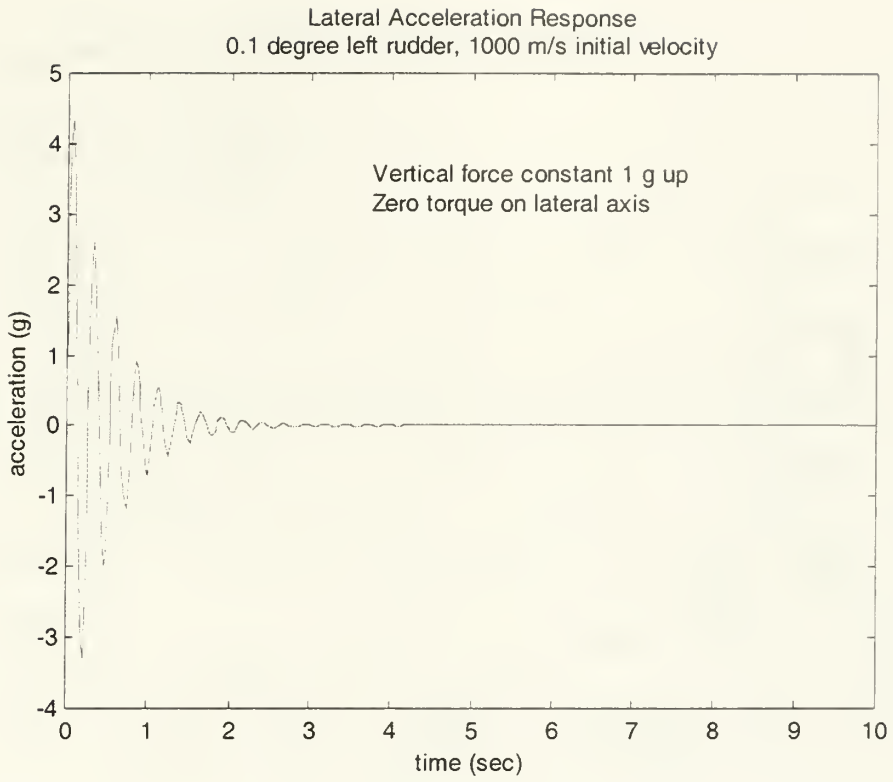


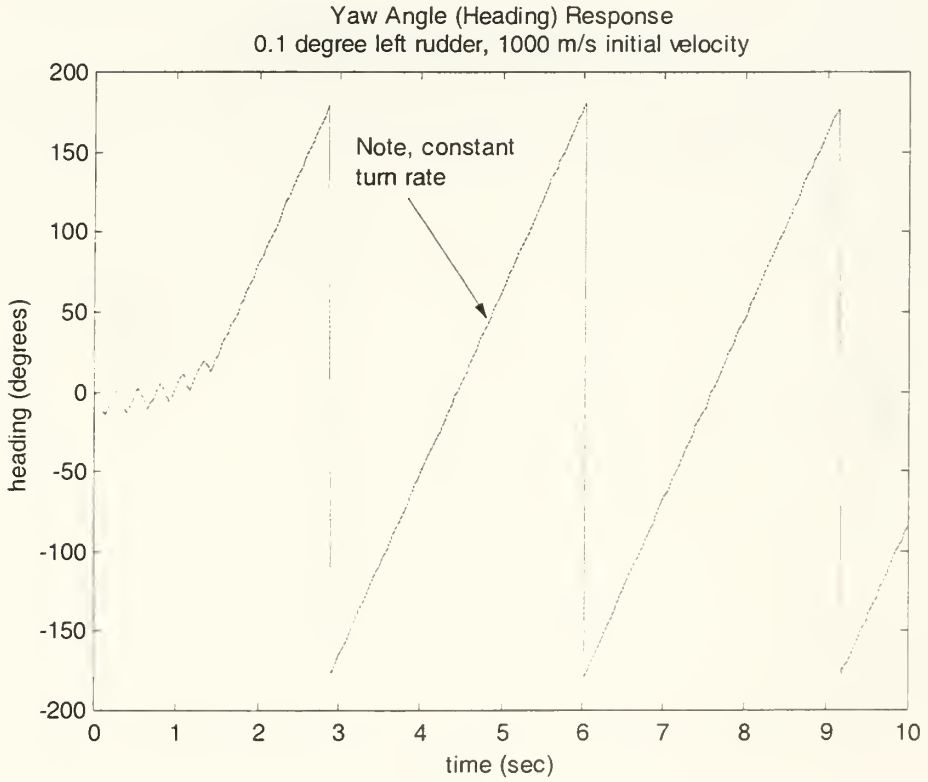

Vertical Acceleration Response
2.25 degree up elevator, 1000 m/s initial velocity
acceleration (g)
time (sec)
Missile operating in open loop
control with steady state input
on elevator deflection angle

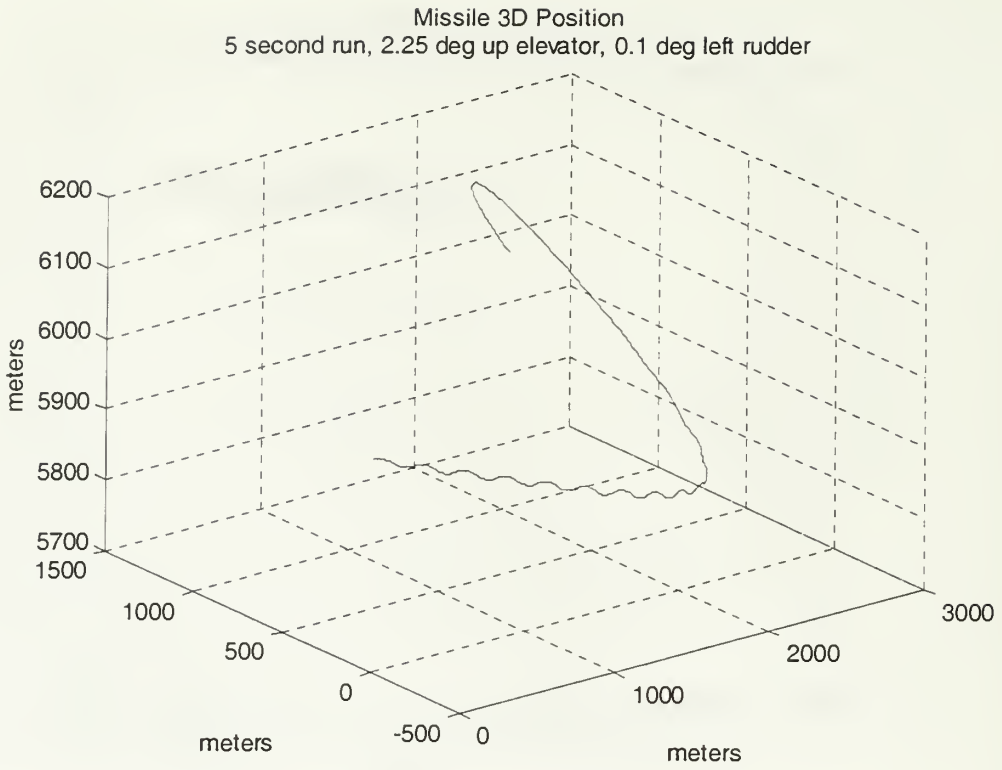

Pitch Angle Response
2.25 degree up elevator, 1000 m/s initial velocity
pitch angle (degrees)
time (sec)
Missile operating in open loop
control with steady state input
on elevator deflection angle

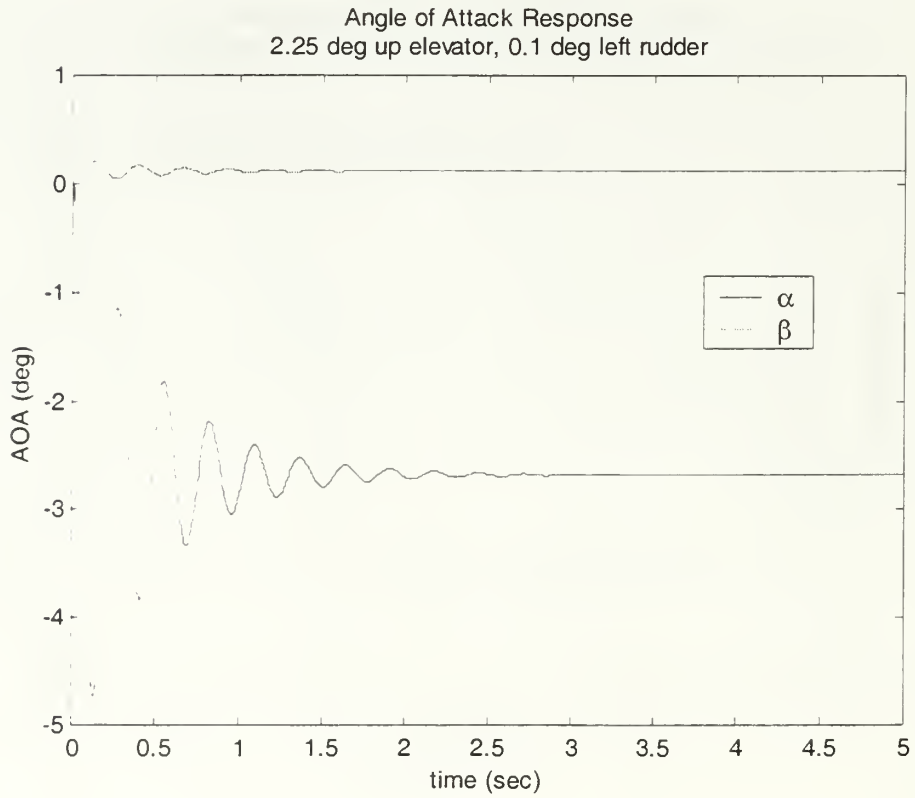

Missile Trajectory
0.1 degree rudder deflection, 1000 m/s initial velocity
Vertical force constant 1 g up
Zero torque on lateral axis
meters
meters
1000
800
600
400
200
0
-200
0
200
400
600
800
1000
1200
1400
1600
1800

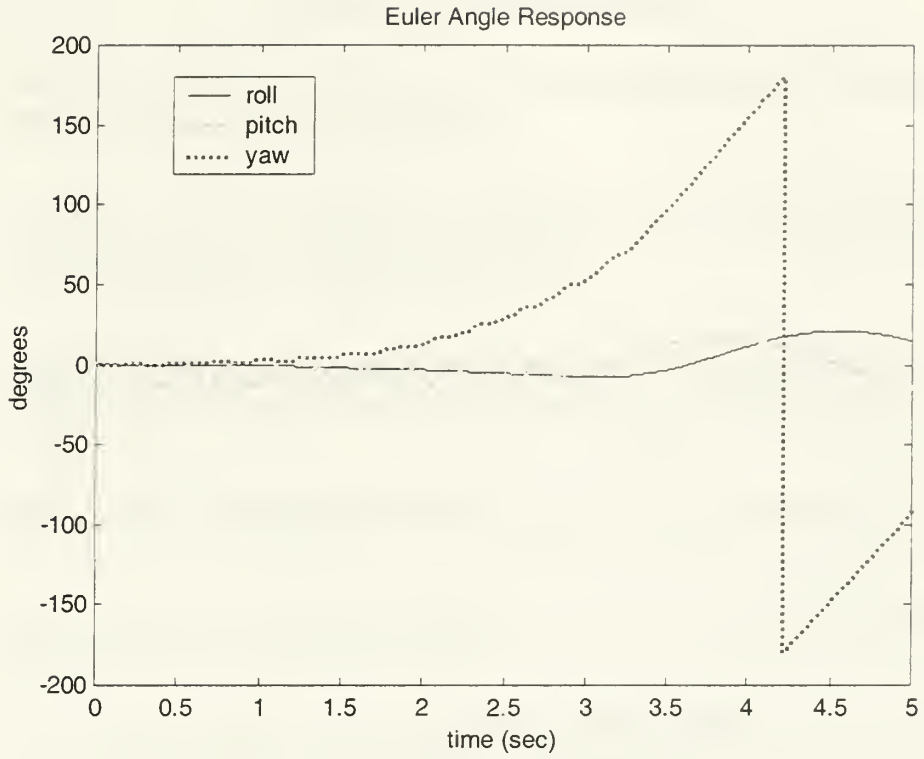

Angle of Attack Response
0.1 degree left rudder, 1000 m/s initial velocity
Vertical force constant 1 g up
Zero torque on lateral axis
AOA (degrees)
time (sec)
0.3
0.25
0.2
0.15
0.1
0.05
0
-0.05
0
1
2
3
4
5
6
7
8
9
10


Lateral Acceleration Response
0.1 degree left rudder, 1000 m/s initial velocity
Vertical force constant 1 g up
Zero torque on lateral axis
acceleration (g)
time (sec)
5
4
3
2
1
0
-1
-2
-3
-4
0
1
2
3
4
5
6
7
8
9
10


Yaw Angle (Heading) Response
0.1 degree left rudder, 1000 m/s initial velocity
Note, constant
turn rate
heading (degrees)
time (sec)
200
150
100
50
0
-50
-100
-150
-200
0
1
2
3
4
5
6
7
8
9
10


Missile 3D Position
5 second run, 2.25 deg up elevator, 0.1 deg left rudder
6200
6100
6000
5900
5800
5700
meters
1500
1000
500
0
-500
meters
0
1000
2000
3000
meters


Angle of Attack Response
2.25 deg up elevator, 0.1 deg left rudder
1
0
-1
-2
-3
-4
-5
AOA (deg)
α
β
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
time (sec)


Euler Angle Response
roll
pitch
yaw
degrees
time (sec)
200
150
100
50
0
-50
-100
-150
-200
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

[1] Westrum, Ron, *Sidewinder, Creative Missile Development at China Lake*, Naval Institute Press, Annapolis, MD, 1999.

[2] Belyakov, R.A. and Marmain,J, *MiG, Fifty Years of Secret Aircraft Design,* Naval Institute Press, Annapolis, MD, 1994.

[3] Ridgely, D. Brett, and McFarland, Michael B., "Tailoring Theory to Practice in Tactical Missile Control," *IEEE Controls Magazine*, December 1999.

[4] Proctor, Paul, "Math Outwits ICBM's?," *Aviation Week*, September 11, 2000.

[5] Stevens, Brian L., and Lewis, Frank L., *Aircraft Control and Simulation,* John Wiley & Sons, Newark, NJ, 1992.

[6] Hutchins, R.G., class notes EC-4330, Navigation, Missile and Avionics Systems, Naval Postgraduate School, Monterey, CA, 1999.

[7] Zarchan, Paul, *Tactical and Strategic Missile Guidance*, Third Edition, American Institute of Aeronautics and Astronautics, Inc., Washington, DC, 1997.

[8] Blakelock, John H., *Automatic Control of Aircraft and Missiles*, Second Edition, John Wiley & Sons, Newark, NJ, 1991.

[9] Sutton, George P., *Rocket Propulsion Elements, An Introduction to the Engineering of Rockets*, Sixth Edition, John Wiley & Sons, Newark, NJ, 1992.

[10] Anderson, John D., *Fundamentals of Aerodynamics*, Second Edition, McGraw-Hill, Inc., Los Angeles, CA, 1991.

[11] Swee, John CS., *Missile Terminal Guidance And Control Against Evasive Targets*, Master's Thesis, Naval Postgraduate School, Monterey, CA, March 2000.

[12] Bryson, Arthur E., and Ho, Yu-Chi, *Applied Optimal Control, Optimization, Estimation, and Control*, Revised Printing, Hemisphere Publishing Corporation, New York, NY, 1975.

[13] Lin, Ching-Fan, *Modern Navigation, Guidance, and Control Processing, Volume II*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1991.

[14] Song, Seong-Ho, and Ha, In-Joong, "A Lyapunov-Like Approach to Performance Analysis of 3-Dimensional Pure PNG Laws," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 30, no. 1, January 1994.

[15] Bar-Shalom, Yaakov, and Li, Xiao-Rong, *Estimation and Tracking: Principles, Techniques, and Software*, YBS, Storrs, MA, 1998.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
   8725 John J. Kingman Road, Suite 0944
   Ft. Belvoir, VA 22060-6218

2. Dudley Knox Library 2
   Naval Postgraduate School
   411 Dyer Road
   Monterey, CA 93943-5101

3. Chairman, Code EC 1
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, CA 93940-5121

4. Professor Robert G. Hutchins, Code EC/Hu 4
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, CA 93940-5121

5. Professor Harold A. Titus, Code EC/Ts 2
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, CA 93940-5121

6. Lieutenant Commander Robert D. Broadston 4
   Operations Department
   U.S.S. Carl Vinson (CVN-70)
   FPO AP 96629-2840

72 290NPG 3094
TH
6/02 22527-200 NLE

DUDLEY KNOX LIBRARY
3 2768 00403382 9